Разработка игры n-Puzzle на основе фреймворка Qt

ЗАДАНИЕ

1. Изучить возможности платформы Qt создания кросс-платформенного программного обеспечения.
2. Изучить средства создания GUI-интерфейсов и связанные технологии.
3. Изучить средства Qt для взаимодействия приложений через сеть на основе сокетов.
4. Разработать игру n-Puzzle.
5. Проверить работоспособность приложения в условиях различных операционных систем.

ЛИТЕРАТУРА

1. Ж. Бланшет, М. Саммерфилд Qt 4: Программирование GUI на C++. — М.: «КУДИЦ-ПРЕСС», 2007.— ISBN 978-5-91136-038-2
2. Макс. Шлее Qt 4: Профессиональное программирование на C++. — СПб.: «БХВ-Петербург», 2007.— ISBN 978-5-9775-0010-6
3. Alan Ezust, Paul Ezust. An Introduction to Design Patterns in C++ with Qt 4 – Prentice Hall, 2006. – ISBN 978-0-13-187905-8
4. Molkentin, Daniel. The Book of Qt 4: The Art of Building Qt Applications. – No Starch Press, Inc., 2006. – ISBN 978-3-937514-12-3
5. Thelin, Johan. Foundation of Qt Development – Apress,2007 – ISBN 978-1-59059-831-3
6. Trolltech Qt 4.5 Whitepaper – Trolltech ASA, 2006.

Введение

В настоящее время наблюдается бурное развитие мобильной связи, платежных систем, интернета в целом, а также появление большого числа устройств со специфическим аппаратным обеспечением и многообразие операционных систем для них. Это породило проблему создания программного обеспечения для целого ряда платформ, то есть была поставлена задача на абстрагирование как от аппаратных, так и от программных платформ, что стало истоком кросс-платформенного программного обеспечения.

В тоже время, рынок диктует важность упрощения интеграции различных приложений, повышения отдачи от инвестиций в ИТ, ускорение и упрощение процесса разработки, что в конечном итоге сводится к задаче снижения стоимости разработки программного обеспечения.

Решение первой из проблем уже найдено и заложено в основе понятия языка программирования, как содержащего стандартизированные конструкции и принципы реализации программного обеспечения в независимости от платформ, на которых ведется разработка. Однако нельзя забывать и про потребности рынка, и диктуемые им правила в отношении упрощения процесса разработки в целях минимизации затрат.

Так со временем, когда на повторное решение уже решенные проблем стало уходить все больше и больше рабочего времени, потребовалось создание более высокоуровневых средств, способных решить обе проблемы. Решение обеих этих проблем является важным этапом будущего развития методов и инструментов разработки программного обеспечения.

Поэтому, Qt, фреймворк (набор библиотек и утилит) программирования на C++, стал следующим шагом в создании более удобного, стабильного и надежного инструментария по созданию переносимых программ с GUI-интерфейсом. Однако данная платформа, как набор классов и утилит, вобрала в себя возможности написания кросс-платформенного программного обеспечения, поддержку работы с базами данных, XML, сетью, 2D и 3D графикой, а также интернационализацию и тестирование.

Цель: изучить платформу разработки прикладного программного обеспечения – Qt и разработать на ее основе игру n-Puzzle для выработки навыков по работе с фреймворком.

Задачи:

1. Изучить возможности платформы Qt создания кросс-платформенного программного обеспечения.
2. Изучить средства создания GUI-интерфейсов и связанные технологии.
3. Изучить средства Qt для взаимодействия приложений через сеть на основе сокетов.
4. Разработать игру n-Puzzle.
5. Проверить работоспособность приложения в условиях различных операционных систем.

Глава 1. Теоретическая часть

1.1. Qt-фреймворк

Qt – фреймворка программирования на C++. Qt поддерживает разработку кросс-платформенных GUI-приложений, основанных на концепции “написал однажды, компилируется везде”. Используя одни и те же исходные коды и простую перекомпиляцию, приложение может быть использовано для Windows 98-XP, Mac OS X, Linux, Solaris, HP-UX и многих других версий Unix с X11. Qt-приложения могут быть откомпилированы для выполнения на встроенных платформах. Qt включает уникальный внутреобъектный механизм взаимодействия под названием “сигналы и слоты”. Qt имеет кросс-платформенную поддержку для 2D и 3D графики, интернационализации, SQL, XML, тестирования. Также предоставляются специфические расширения приложений для конкретных платформ.

Приложения Qt может быть созданы с помощью Qt Designer, гибкого инструмента для создания пользовательского интерфейса с поддержкой взаимодействия с интегрированной средой разработки (IDE), и с помощью Qt Creator, включающего в себя Qt Designer, легковесная кросс-платформенная среда разработки, заточенная для разработки под C++ и Qt.

Набор инструментов Qt реализован на следующих платформах:

- Qt/X11 — Qt для X Window System
- Qt/Mac — Qt для Apple Mac OS X
- Qt/Windows — Qt для Microsoft Windows
- Qt/Embedded — Qt для встраиваемых платформ (PDA, Smartphone, ...)

Для каждой из перечисленных платформ доступны четыре редакции Qt:

- Qt Console — редакция для разработки приложений без графического интерфейса.
- Qt Desktop Light — редакция для начального уровня GUI, из которой исключены возможности по разработке сетевых приложений и работа с базами данных.
- Qt Desktop — полная редакция.
- Qt Open Source Edition — полная редакция, для разработки программ с открытым исходным кодом.

Все редакции поддерживаются большим количеством компиляторов, куда входит и GCC C++ компилятор, а в случае с Qt/Windows – Visual Studio.

Нововведениями Qt являются следующие ключевые принципы/концепции:

- полная абстракция GUI. Qt использует собственный движок для рисования и собственные элементы управления, эмитируя при этом ту платформу, на которой работает. Эта особенность делает проекты легко переносимыми, поскольку очень немногие классы Qt зависят от целевой платформы.
- мета-объектный компилятор. Известный как moc, этот инструмент запускается перед компиляцией Qt программы и генерирует мета-информацию о классах, используемых в программе. В дальнейшем эта мета-информация используется Qt для обеспечения работы возможностей, отсутствующих в стандарте C++. Таковой, например, является сигнально-слотовый механизм.

Мной использовалась редакции для платформ Qt/X11 и Qt/Windows. Так как для изучения выбранного фреймворма мне необходимо максимум функциональности за минимум вложений средств, а исходные коды не предполагается скрывать от посторонних глаз, мной была выбрана редакция Qt Open Source Edition.

1.2. Makefile, qmake и Project-файлы

C++ приложения включает множество файлов исходных кодов, заголовочных файлов и внешних библиотек. В течение нормального процесса разработки проекта, исходные файлы и библиотека добавляются, изменяются и удаляются. В процессе тестирования, разработки проект пересобирается множество раз. Для того чтобы сделать исполняемыми, все тексты исходных кодов, должны быть перекомпилированы, и все объектные файлы должны быть перелинкованы.

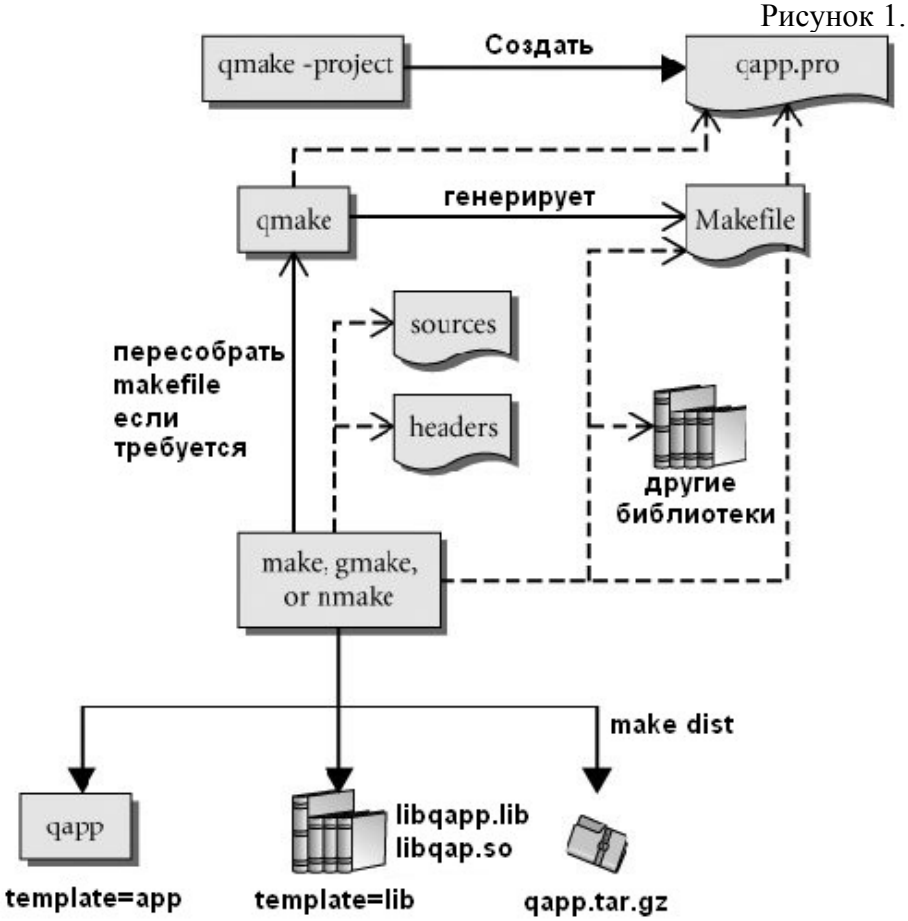
Наиболее широко используемой утилитой для выполнения практической работа по пересборке проекта является make. Эта утилита читает описание проекта и инструкции для компиляции для компилятор из файла под названием Makefile, который представляет собой скрипт для командного интерпретатора и включает (как минимум):

- правила для сборки определенных типов файлов (например, для того чтобы получить .o-файлы из .cpp, необходимо выполнить gcc –c для .cpp-файла)
- цели, которые определяют какие исполняемые файлы и/или библиотеки должны быть собраны.
- зависимости, представляющие собой список целей, которые необходимо пересобрать в случае изменения определенных файлов.

Команда make по умолчанию загружает файл под названием Makefile из текущего рабочего каталога и выполняет определенные шаги по сборке (компиляция и линковка).

Особенностью использования утилиты make является то, что пересобираются только файлы, которые должны быть пересобраны, вместо перекомпиляции каждого исходного файла каждый раз. Ниже представлена диаграмма, показывающая шаги, выполняемые при сборке приложения Qt.

Рисунок 1.



С Qt нет необходимости писать Makefile. Qt предоставляет утилиту под названием qmake для генерации Makefile. Но все еще необходимо знать, как запускать make, и понимать

сообщения, которые она выводит. Многие IDE запускают make (или другую похожую утилиту) “за кулисами” и выводят или фильтруют его вывод.

Следующая последовательность команд показывает, какие файлы создаются на каждом шаге процесса сборки.

```
src/qapp> ls -sF
total 296
  4 main.cpp
src/qapp> qmake -project
src/qapp> ls -sF
total 296
  4 main.cpp  4  qapp.pro
src/qapp> qmake
src/qapp> ls -sF
total 296
  4 main.cpp  4 qapp.pro   4 Makefile
src/qapp> make
g++ -c -pipe -g -Wall -W            # compile step
    -D_REENTRANT -DQT_CORE_LIB -DQT_GUI_LIB -DQT_SHARED
    -I/usr/local/qt/mkspecs/linux-g++ -I.
    -I/usr/local/qt/include/QtGui -I/usr/local/qt/include/QtCore
    -I/usr/local/qt/include -I. -I. -I.
    -o main.o main.cpp
g++ -Wl,-rpath,/usr/local/qt/lib     # link step
    -L/usr/local/qt/lib -L/usr/local/qt/lib -lQtGui_debug
    -L/usr/X11R6/lib -lpng -lXi -lXrender -lXinerama -lfreetype
    -lfontconfig -lXext -lX11 -lm -lQtCore_debug -lz -ldl -lpthread
    -o qapp main.o
src/qapp> ls -sF
total 420
  4 main.cpp    132 main.o   124 qapp*
  8 Makefile    4 qapp.pro
src/qapp>
```

Следует отметить, что существует возможность видеть аргументы, посылаемые компилятору при выполнении команды make. Если какие-либо ошибки появятся, их невозможно будет не заметить.

Команда make проверяет зависимости и выполняет каждый шаг сборки, определенный в Makefile.

Make может удалить сгенерированные файлы с использованием двух целей clean и distclean. Различия в их действии представлены ниже.

```
src/qapp> make clean
rm -f main.o
rm -f *~ core *.core
src/qapp>  ls
 Makefile  main.cpp  qapp  qapp.pro
src/qapp> make distclean
rm -f qmake_image_collection.cpp
rm -f main.o
rm -f *~ core *.core
rm -f qapp
rm -f Makefile
src/qapp> ls
main.cpp  qapp.pro
```

После make distclean остаются только исходные файлы, которые могут быть помещены в tar-архив для дальнейшего распространения, например, через Internet.

Использование qmake

Утилита qmake обеспечивает проектно-ориентированную систему для управления процессом сборки приложений, библиотек и других компонентов. Этот подход дает разработчикам возможность задавать настройки для сборки всего в одном файле проекта (.pro-файл).

Файл проекта включает всю информацию, требуемую qmake для сборки приложений, библиотек или плагинов. Любой файл проекта используется для задания переменных, определяющих исходные и заголовочные файлы, используемые в проекте. Более сложный файл проекта включает конструкции, используемые для управления процессом сборки.

qmake читает определенные переменные в каждом файле проекта и использует их значения для определения Makefile'a. Ниже приведен список переменных, которые qmake использует.

- SOURCES (список исходных кодов, используемых для сборки проекта)
- HEADERS (список заголовочных файлов, используемых при сборке проекта)
- FORMS (список .ui-файлов)
- CONFIG (общие настройки проекта)
- QT (специфические опции Qt)
- DESTDIR (папка, в которую разместятся exe или бинарные файлы)
- TARGET (имя итогового исполняемого файла)
- RESOURCES (список файлов ресурсов (.rc), которые будут включены в финальный проект)
- TEMPLATE (шаблон проекта, определяющий будет ли выходной файл приложением, библиотекой или плагином)

Также есть возможность добавлять комментарии в файл проекта. Комментарии выглядят следующим образом:

```
# Это комментарий
```

qmake предоставляет некоторое количество встроенных функций. Большинство используемых в любом проекте функций используют имя файла, как аргумент. Функция include() наиболее часто используется для включения других файлов проектов.

```
include(other.pro)
```

Есть возможность определения условия использования тех или иных файлов в зависимости от платформы, например, как в следующем примере.

```
win32 {
    SOURCES += paintwidget_win.cpp
}
```

Действие по добавлению еще одного файла выполняется, если переменная win32 определена или в случае запуска qmake с опцией командной строки –win32.

Следующий код демонстрирует использование простого цикла по элементам списка.

```
EXTRAS = handlers tests docs
for(dir, EXTRAS){
    exists($$dir){
      SUBDIRS += $$dir
    }
}
```

Помимо вышеперечисленных функций существуют многие другие, дающие возможность манипулировать строками и путями, поддерживать пользовательский ввод и вызывать внешние утилиты. Полный список функций доступен в руководстве по qmake.

Переменная TEMPLATE, задающая тип создаваемого проекта, может принимать любое из представленных ниже значений.

Шаблоны проектов

- app (по-умолчанию, создает Makefile для сборки приложения)
- lib (создает Makefile для сборки библиотеки)
- subdirs (создает Makefile, включающий правила для подкаталогов, каждый подкаталог должен включать свой файл проекта)
- vcapp (создает файл проекта по созданию приложения для Visual Studio)
- vclib (создает файл проекта по созданию библиотеки для Visual Studio)

Переменная CONFIG определяет особенности того, как будет запускаться компилятор, и может принимать ниже перечисленные значения.

- release (проект создается в режиме релиза, но опция игнорируется, если debug также определена)
- debug (проект создается в режиме отладки)
- debug_and_release (проект собирается в обоих режимах)
- build_all (если опция debug_and_release определена, проект собирается по-умолчанию в обоих режимах)
- ordered (когда используется шаблон subdirs, эта опция определяет порядок обработки списка каталогов)
- warn_on (компилятору следует выводить настолько много сообщений о предупреждениях, насколько это возможно. Данная опция игнорируется, если warn_off уже была определена)
- warn_off (компилятор пытается выводить наименьшее число предупреждений).

Более полная информация о способах использования qmake, описание всех возможных переменных и функциях, а также способах настройки окружения может быть найдена в руководстве по Qt. Для создания и работы с небольшими проектами выше изложенной информации вполне достаточно. Несколько примеров файлов проектов, созданных мной при изучении библиотеки Qt, приведены в Приложении 2.

1.3. Модули Qt

Иерархия классов Qt имеет четкую внутреннюю структуру.

Qt 4 – это библиотека, состоящая из более маленьких библиотек (модулей). Наиболее популярные модули:

- QtCore (QObject, QThread, QFile и многие другие классы, не связанные с графическим интерфейсом)
- QtGui (все классы, наследуемые от QWidget и некоторые связанные классы для программирования графического интерфейса)
- QtNetwork (классы, используемые для программирования взаимодействия по сети и передачи данных между хостами по специфическим протоколам, т.е. http, tcp, udp)
- QtOpenGL (программирование графики OpenGL)
- QtSql (взаимодействие с базами данных)
- QtSvg (работа с SVG, масштабируемая векторная графика)
- QtXml (для преобразования и сериализации в формате XML, включена поддержка XML, SAX- и DOM-классов)
- Qt3Support (классы для совместимости с предыдущей библиотекой Qt)
- QtScript (включает поддержку языка сценария)
- QtTest (классы для тестирования)

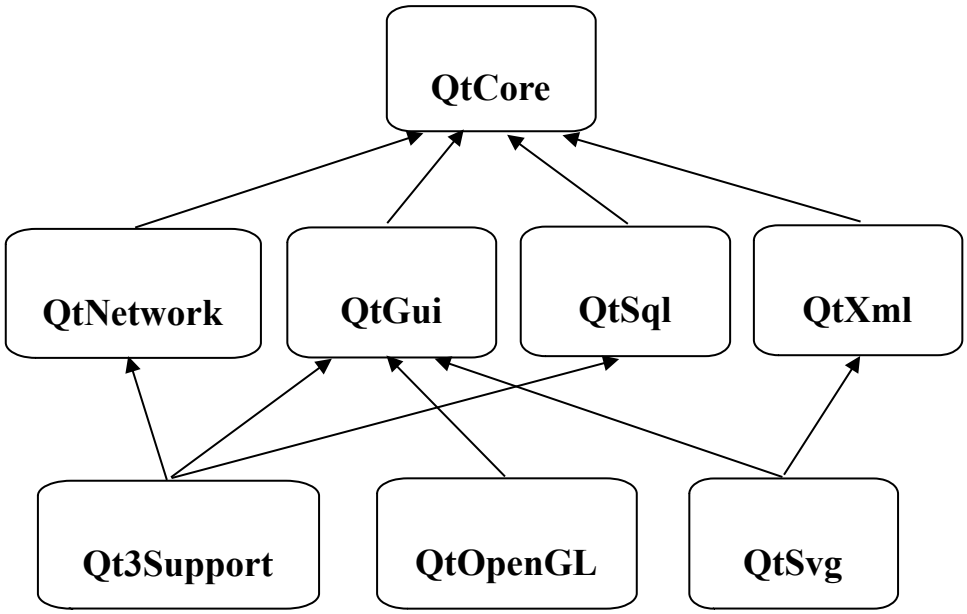


Рисунок 2.

Все модули за исключением QtCore и QtGui необходимо прописать в файле проекта qmake и для того чтобы ими пользоваться. Например,

```
QT += xml  # для использования xml модуля
QT += sql  # для использования SQL модуля
QT -= gui  # для исключения модуля GUI из проекта
```

1.4. Модуль QtCore

Этот модуль является базовым для приложений и не содержит классов, относящихся к интерфейсу пользователя. Для реализации консольного приложения можно ограничиться одним этим модулем.

Наиболее важные классы этого модуля:

- контейнерные классы QList, QVector, QMap;
- классы для ввода и вывода QIODevice, QTextStream, QFile;
- классы процесса QProcess и для программирования многопоточности QThread, QWaitCondition, QMutex;
- классы для работы с таймером QBasicTimer и QTimer;
- классы для работы с датой и временем QDate и QTime;
- класс QObject, являющийся краеугольным камнем объектной модели Qt;
- базовый класс событий QEvent;
- класс для сохранения настроек приложения QSettings;
- класс приложения QCoreApplication, из объекта которого, если требуется, можно запустить цикл событий.

Наиболее важным классом модуля QCore является QCoreApplication. Объект класса приложения QCoreApplication содержит объекты, подсоединенные к контексту операционной системы. Объект класса QCoreApplication существует втечении работы всего приложения, должен создаваться в приложении только один раз и остается доступным в любой момент работы программы. В назначение этого объекта входит:

- управление событиями приложения и операционной системы;
- передача и предоставление аргументов командной строки.

1.4.1. Контейнеры

Классы контейнеров Qt используются для объединения типов-значений, включая указатели на объектные типы (но не сами объектные типы сами по себе). Контейнеры в Qt обозначены как шаблоны классов. Каждая структура данных оптимизирована для различного рода операций. В Qt 4 собрано несколько шаблонов классов контейнеров:

- QList<T> (реализуется с использованием массива, в котором границы могут изменять с обоих концов; шаблон сделан для оптимизации доступа по случайному индексу, для списков длиной менее тысячи элементов, также дает хорошую производительность при использовании в операциях prepend() и append());
- QStringList (класс созданный для удобства, наследуется от QList<QString>, содержит встроенные функции для работы со строками);
- QLinkedList<T> (оптимизирован для последовательного доступа с помощью итераторов и обеспечивает быструю, постоянную по времени вставку по всему списку, сортировка и поиск происходит медленно, имеет несколько удобных функций для часто используемых операций);
- QVector<T> (хранит данные в смежных областях памяти и оптимизирован для случайному доступу по индексу; в общем случае, создается с определенным размером; вставка, добавление в конец/начало обходится дорого);
- QStack<T> (наследован как public от QVector<T>, поэтому интерфейс QVector<T> также доступен и для объектов класса QStack, однако, концепция “последний вошел – первый вышел” реализуется функциями push(), pop() и top());
- QMap<Key, T> (упорядоченный ассоциативный контейнер, который сохраняет пары (ключ, значение) и спроектирован для быстрого поиска значений, ассоциированных с ключом для легкой вставки; ключи содержатся отсортированными; для ключа должны быть определены операции operator<() и operator==());
- QHash<Key, T> (ассоциативный контейнер, использующий хэш-таблицы для облегченного поиска и вставки, но медленного поиска и не сортируется; для ключа должен быть определен оператор operator==());
- QMultiMap<Key, T> (класс-наследник от QMap и QHash, вследствие чего появилась возможность ассоциировать множество значений с одним ключом);
- QCache<Key, T> (ассоциативный контейнер, но обеспечивает наивысшую скорость доступа к недавно использовавшимся элементам с автоматическим удалением редко использовавшихся на основании стоимостной функции);
- QSet<T> (хранит значения типа T, используя QHash, с ключами типа T и псевдозначениями, ассоциированными с каждым ключом; такая договоренность оптимизирует поиск и вставки; QSet имеет несколько функций для работы с множествами (объединение, пересечение, вычитание и др.); конструктор по-умолчанию создает пустое множество).

Параметр типа T для шаблона класса контейнера или тип ключа для ассоциативного контейнера должен быть типом-значением, другими словами, должен иметь пустой конструктор, копирующий конструктор и оператор присваивания.

Допускаются базовые типы (т.е. int, double, char и др.). Также допускаются некоторые типы Qt (т.е. QString, QDate, QTimer). QObject и типы, наследованные от QObject, использовать запрещается. При необходимости сгруппировать объекты некоторого недопустимого типа есть возможность определить контейнер указателей (QList<QFile*>).

Итераторы и алгоритмы

Контейнеры могут обрабатываться с помощью итераторов, реализованных в самой Qt. Qt также включает набор базовых алгоритмов для работы с контейнерами.

Существует два типа итераторов:

- в стиле Java
- в стиле STL

Итераторов в стиле Java представляют собой объекты, а не указатели. Их использование, если сравнивать с итераторами в стиле STL делает код более компактным, однако их использование заметно увеличивает размер получаемого в итоге исполняемого модуля.

Итераторы в стиле STL могут быть использованы совместно с алгоритмами STL. На данные момент в Qt реализована только часть возможностей STL. Хотя в будущем ожидается, что возможности Qt в плане работы с контейнерами приблизится к возможностям STL. В тоже время ничего не запрещает одновременно использовать эти два инструмента одновременно. Единственная проблема, которая здесь появляется, это отсутствие для STL такого ставшего уже привычным механизма автодополнения в программах-редакторах исходных кодов, что требует от программистов более детального знания этой библиотеки.

Перебор

Если требуется обрабатывать в цикле списки, следует использовать макрос foreach (variable, containter). Макрос работает со всеми контейнерами Qt. Например, чтобы пробежаться по всем элементам QStringList, можно воспользоваться следующим кодом:

```
foreach (QString value, valueList)
    doSomething (value);
```

Когда требуется бесконечный цикл, в большинстве случаев используют пустой цикл for (for(;;)…) или часто используемый цикл while (while(true)…). Чтобы сделать код более легким для чтения, можно использовать макрос forever. Следующие строчки показывают, как это может выглядеть на практике:

```
forever
    doSomething();
```

Для исключения из глобального пространства имен ключевых слов foreach и forever следует добавить no_keywords к переменной CONFIG из project-файла. После чего макросами foreach и forever все еще есть возможность пользоваться по именам Q_FOREACH и Q_FOREVER.

1.4.2. QString

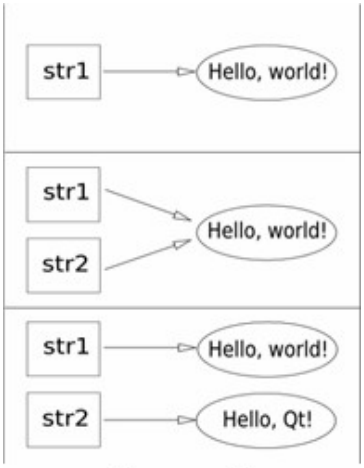


Рисунок 3.

Однако Qt также включает множество классов, которые не наследуются от QObject, так как они не требуют ни сигналов и слотов, ни автоматического управления памятью. Эта групп классов включает, например, один из наиболее важных классов, QString, который отвечает за строки. Строки в QString сохраняются и обрабатываются в Unicode-формате, то есть поддерживаются наборы символов West European, Cyrillic, Arabic, Hebrew, Chinese и многие другие. По этой причине, Qt может использоваться очень эффективно в программах, имеющих дело с самыми различными языками

Классы QImage (используется для загрузки/сохранения изображений), QColor (хранит цвет) и многие другие, а также не наследуемые от QObject классы.

Библиотека Qt гарантирует, что когда эти классы используются, два экземпляра никогда не имеют одинакового содержимого. Например, будет расточительно иметь несколько копий строк вместо всего одной копии, используемой всеми клиентскими объектами, поэтому Qt включает специальный планировщик для уничтожения неиспользуемых копий.

1.4.3. QObject

Важный класс, от которого наследуются, в том числе все классы графического интерфейса – QObject.

```
class QObject {
  public:
    QObject(QObject* parent=0);
    QObject * parent () const;
    QString objectName() const;
    void setParent ( QObject * parent );
    const ObjectList & children () const;
    // ... и так далее ...
};
```

Важной особенностью объектов класса QObject является то, что они никогда не могут быть переданы по значению в какую-либо функцию. Это связанно с тем, что копирующий конструктор объявлен не как public. Это не значит, что объекты класса QObject не могут быть скопированы. Передача информации между объектами все еще возможно, но любые объекты остаются уникальными.

Каждый объект класса QObject может иметь (в большинстве случаев имеет) одного родительский объект и произвольно большое количество QObject* на детей. Каждый QObject сохраняет указатели на своих детей в QObjectList. QObjectList – это синоним для QList<QObject*>. Так как каждый ребенок – это QObject и может иметь неограниченно большое количество своих детей, легко видеть, почему копирование объектов класса QObject запрещено.

Каждый объект класса QObject – родитель, управляющий своими наследниками. Это означает, что деструктор класса QObject автоматически удаляет у своего объекта всех наследников. Поэтому следует избегать одной из самых распространенных ошибок программистов на C++ при программировании с использованием Qt – это самостоятельный контроль процесса выделения/освобождения памяти для объекта и нединамическое создание элементов управления.

При программировании с Qt важно помнить, что все объекты должны создаваться в памяти динамически, с помощью оператора new. Исключение из этого правила могут составлять только объекты, не имеющие предков.

Для получения информации об объектной иерархии существует два метода: parent() и children(). С помощью метода parent() можно определить объект-предок. Для объекта верхнего уровня этот метод вернет значение 0. Метод children() возвращает константный указатель на список QObjectList объектов-потомков. Для расширенного поиска существует метод findChildren(), возвращающий список указателей на объекты. Параметр этого метода не обязательный, им может быть либо строка, либо регулярное выражение, а вызов метода без параметров приведет к тому, что он вернет список указателей на все объекты. Поиск производится рекурсивно.

1.5. Методы отладки

Для отладки программы полезен метод Object::dumpObjectInfo(), который показывает следующую информацию, относящуюся к объекту:

- имя объекта;
- класс, от которого был произведен объект;
- сигнально-слотовые соединения.

Вся эта информация поступает в стандартный поток вызова stdout.

Также при отладке можно воспользоваться методом Object::dumpObjectTree(), предназначенным для отображения объектов-потомков в виде иерархии.

В заголовочном файле QtGlogal содержатся определения двух макросов: Q_ASSERT() и Q_CHECK_PTR().

- Q_ASSERT() принимает, в качестве аргумента, значение булевого типа и выводит предупреждающее сообщение, если это значение не равно true.
- макрос Q_CHECK_PTR() принимает указатель и выводит предупреждающее сообщение, если переданный указатель равен 0, а это означает, что указатель либо не был инициализирован, либо операция по выделению памяти прошла неудачно.

Qt предоставляет глобальные функции qDebug(), qWarning(), qFatal(), которые также определены в заголовочном файле QtGlobal. В эти функции передаются форматированная строка и различные параметры. В Microsoft Visual Studio вывод этих функций производится в окно отладчика, а в ОС Linux – в стандартный поток вывода ошибок. Вызов функции qFatal() после вывода сообщения сразу завершает работу всего приложения.

Использование qDebug() является самым простым и напоминает использование стандартного объекта потока вывод в C++ cout. Например, вывести сообщение в отладчике или на консоли с помощью функции qDebug() можно следующим образом:

qDebug() << “Test”;

Эта функция создает объект класса потока QDebug, передавая в его конструктор аргумент QtDebugMsg. Можно было бы, конечно, поступить и так:

QDebug(QtDebugMsg) << “Test”;

В тоже время предыдущий пример выглядит более компактно, поэтому рекомендуется пользоваться именно им.

1.6. Типы данных

Qt предоставляет целый набор типов, определенных как кросс-платформенные. Это означает, что беззнаковое 16-битное число остается таковым на всех платформах.

Qt также предоставляет переменный объект, который можно использовать для конвертации данных между несколькими типами.

Переменный тип

Тип QVariant может быть использован для хранения большинства типов, используемых в приложениях Qt. При определении значения как объекта QVariant, значение автоматически преобразуется к типу QVariant. Для преобразования QVariant в другой тип, следует задать ожидаемый тип. Все типы, доступные в модуле QtCore могут быть получены преобразованием с использованием методов to*Type*, где *Type* – имя типа. Поддерживаются следующие типы: bool, QByteArray, QChar, QDate, QDateTime, double, int, QLine, QLineF, QList<QVariant>, QLocale, qlonglong, QMap<QString, QVariant>, QPoint, QPointF, QRect, QRectF, QRegExp, QSize, QSizeF, QString, QStringList, QTime, uint, qulonglong и QUrl.

Типы по размеру

Следующие типы имеют точно тот размер, что они обозначают. При использовании потоков Qt при чтении и записи, границы этих типов также остаются постоянными при смене платформы.

- quint8 (8-битное беззнаковое число в промежутке 0–255)
- quint16 (16-битное беззнаковое число в промежутке 0–65535)
- quint32 (32-битное беззнаковое число в промежутке 0–4294967295)
- quint64 (64-битное беззнаковое число в промежутке 0–1.844674407e19)
- qint8 (8-битное знаковое число в промежутке -128–127)
- qint16 (16-битное знаковое число в промежутке -32768–32767)
- qint32 (32-битное знаковое число в промежутке -2147483648–2147483647)
- qint64 (64-битное знаковое число в промежутке -9.223372036e18–9.223372036e18)

При использовании qreal нет гарантии в постоянстве размера, потому что qreal предоставляет значение с двойной точностью (double) на всех платформах за исключением ARM. На ARM платформах этот тип представляется как float.

1.7. Ресурсная система

В Qt есть возможность доступа не только к внешним устройствам, а также к предварительно внедренным в исполняемый модуль приложения двоичным данным или тексту. Это обеспечивается ресурсной системой Qt.

Ресурсы преобразуются в программный код C++ ресурсным компилятором Qt (rcc). Можно указать qmake, что необходимо включить специальные правила для выполнения rcc, добавляя следующую строку в файл .pro:

RESOURCES = myresourcefile.qrc

Фактически myresourcefile.qrc – это XML-файл, который содержит список файлов, внедренных в исполняемый модуль, например, файлы изображений или иконок.

```
<RCC>
<qresource prefix="/images" >
    <file>images/qtcreator_logo_16.png</file>
    <file>images/example.jpg</file>
</qresource>
</RCC>
```

В приложении ресурсы опознаются по префиксу пути :/. В этом примере файл example.jpg имеет путь :/images/images/example.jpg и может быть прочитан как любой файл, используя QFile.

Преимуществом внедрения данных в исполняемый модуль является невозможность их потери и возможность создания действительно автономных исполняемых модулей (если использовать статическую компоновку). Недостатками являются необходимость замены всего исполняемого файла при изменении внедренных данных и увеличение размера исполняемого модуля из-за дополнительного расхода памяти под внедренные данные.

Ресурсная система Qt обладает дополнительными возможностями, включая поддержку псевдонимов файлов и локализации.

1.8. Модуль QtGui

Модуль содержит более 500 классов, используемых для программирования графического интерфейса пользователя. Модуль QtGui дополняет QtCore функциональностью GUI.

Для включения определений классов обоих этих модулей, используйте следующую директиву:

#include <QtGui>

Модуль QtGui является частью Qt Desktop Light Edition, Qt Desktop Edition и Qt Open Source Edition.

1.8.1. QApplication и обработчик событий

Интерактивные Qt-приложения с графическим пользовательским интерфейсом имеют различие в своей организации от консольных приложений, так как они основаны на событиях и часто многопотоковые. Один поток для ответов на пользовательские запросы и еще один или несколько других потоков для выполнения ресурсозатратных действий: чтение больший файлов или выполнение сложных вычислений.

Объекты части посылают сообщения друг другу. Объекты могут посылать информацию друг другу различными способами. Элементы графического интерфейса посылают сообщения другим QObject'ам в ответ на пользовательские действий такие как щелчки мыши и события клавиатуры. Элементы графического интерфейса также отвечают на события, такие как перерисовка, изменение размера или закрытие. Более того, объекты класса QObject могут передавать информацию друг другу, используя сигналы и слоты.

Очередь событий – это программная структура, позволяющая отправлять сообщения, организовывать очередь из них и выставлять приоритеты.

Пример ниже демонстрирует простое приложение, инициализирующее цикл обработки событий вызовом метода QApplication::exec().

```
int main(int argc, char * argv[]) {
   QApplication myapp(argc, argv);        // <-- 1
   QWidget rootWidget;
   setGui(&rootWidget);
   rootWidget.show();                     // <-- 2
   return myapp.exec();                   // <-- 3
};
```

(1) Каждое GUI, многопотоковое или основанное на событиях Qt-приложение должно иметь объект класса QApplication, определенный в самом начале функции main().

(2) Показ виджета на экране.

(3) Вход в цикл обработки сообщений.

Фактически функция QApplication::exec() проставляет собой не более чем вызов функции processEvents() в цикле while. Использование функции processEvents() может быть очень полезное в случаях, когда генерируется события, для обработки которого требуется слишком много времени. В этом случае интерфейс пользователя становится невосприимчивым к действиям пользователя. Например, любые сгенерированные оконной системой события (перерисовка приложения в том числе) во время сохранения файла на диск не будут обрабатываться до тех пор, пока весь файл не будет записан.

Одно из решений заключается в применении многопоточной обработки: один поток для работы с интерфейсом пользователя и другой процесс для выполнения операции сохранения файла (или любой другой длительной операции).

Более простое решение заключается в выполнении частых вызовов функции QApplication::processEvents(). Ниже приводится пример сохранения работоспособности интерфейса пользователя при помощи функции processEvents().

```
bool MainWindowImpl::writeFile(const QString &fileName)
{
    QFile file(fileName);
    …
    for (int row = 0; row < RowCount; ++row) {
        for (int column = 0; column < ColumnCount; ++column) {
            QString str = formula(row, column);
            if (!str.isEmpty())
                out << quint16(row) << quint16(column) << str;
        }
        qApp->processEvents();
    }
    return true;
}
```

Однако существует опасность того, что пользователь может закрыть окно во время выполнения операции сохранения файла или даже вызвать повторное выполнение данной функции, что приведет к непредсказуемым результатам. Наиболее простое решение заключается в замене вызова

```
qApp->processEvnets();
```

на вызов

```
qApp->processEvents(QEventLoop::ExludeUserInputEvents);
```

которое указывает Qt на необходимость игнорирования событий мыши и клавиатуры.

1.8.2. Сигналы и слоты

После того как главный поток C++ программы вызывает qApp->exec(), он входит в цикл обработки событий. Во время его работы появляется возможность для объектов с классами, унаследованными от QObject, посылать сообщения друг другу.

Сигнал – это сообщение, предоставляющее в классе описание функции, возвращающей void. Сигнал имеет список параметров, но не имеет реализации, т.к. эту работу принимает на себя MOC. Из этого следует, что не имеет смысла определять сигналы как private, protected или public, так как они играют роль вызывающих методов. Сигнал является частью интерфейса класса и выглядит как функция, должна вызываться объектом этого же класса. Вызвать сигнал можно при помощи ключевого слова emit. Ввиду того, что сигналы играют роль вызывающих методов, конструкция высылки сигнала emit doIt() приведет к обычному вызову метода doIt().

Слот – это функция-член, возвращающая void. Слот может быть вызван, как и обычный метод, поэтому определяются в классе как public, private или protected. Соответственно, перед каждой группой слотов должно стоять: private slots:, protected slots: или public slots:. Слоты могут быть и виртуальными. По данным фирмы Trolltech, соединение сигнала с виртуальным слотом примерно в десять раз медленнее, чем с невиртуальным. Но есть небольшие ограничения, отличающее обычные методы от слотов: в слотах нельзя использовать параметры по умолчанию и определять слоты как static.

Сигнал одного объекта может быть связан со слотами одного или нескольких других объектов. Синтаксис функции connect() приведет ниже.

```
bool QObject::connect(senderqobjptr, SIGNAL(signalname(argtypelist)),
                      receiverqobjptr, SLOT(slotname(argtypelist)),
                      optionalConnectionType);
```

senderqobjptr – указатель на объект, высылающий сигнал;

signal – это сигнал, с которым осуществляется соединение. Прототип (имя аргумента) метода сигнала должен быть заключен в специальном макросе SIGNAL(method());

receiver – указатель на объект, который имеет слот для обработки сигнала;

slot – слот, который вызывается при получении сигнала. Прототип слота должен быть заключен в специальном макросе SLOT(method());

optionalConnectionType – управляет режимом обработки. Имеется три возможных значения: Qt::DirectConnection – сигнал обрабатывается сразу, вызовом соответствующего слота, Qt::QueuedConnection – сигнал преобразуется в событие и становится в общую очередь для обработки, Qt::AutoConnection – это автоматический режим, который действует следующим образом: если объект, высылающий сигнал, находится в одном потоке с объектом, принимающим его, то устанавливается режим Qt::QueuedConnection, иначе – режим Qt::DirectConnection. Режим Qt::AutoConnection установлен в методе connect() по умолчанию.

В случае, когда слот содержится в классе, из которого производится соединение, то можно воспользоваться сокращенной формой метода connect(), опустив третий параметр, указывающий на объект-получатель. Другими словами, если в качестве объекта-получателя должен стоять указатель this, его можно просто не указывать.

Высылку сигналов заблокировать можно на некоторое время, вызвав метод Object::blockSignals() с параметром true. Объект будет «молчать», пока блокировка не будет снята тем же методом Object::blockSignals() с параметром false. При помощи метода Object::signalsBlocked() можно узнать о текущем состоянии блокировки сигналов.

Любой QObject, имеющий сигнал, может вызвать его. Это приведет к косвенному вызову всех присоединенных слотов. QWidget уже вызывает сигналы в ответ на события, так что при написании программы только нужно создать правильные связи для получения этих сигналов. Аргументы, посылаемые при использовании выражения emit доступны как параметры в функциях слотов, подобно вызову функции, за исключением того, что вызов происходит не напрямую. Список аргументов – это способ передачи информации от одного объекта другому.

Механизм сигналов и слотов полностью замещает старую модель функций обратного вызова, он очень гибок и полностью объектно-ориентирован. Сигналы и слоты – это ключевой концепт программирования с помощью Qt, позволяющий соединить вместе объекты, несвязанные друг с другом. Эта особенность идеально вписывается в концепцию объектной ориентации и не противоречит человеческому восприятию.

Использование механизма сигналов и слотов дает программисту следующие преимущества:

- каждый класс, унаследованный от QObject, может иметь любое количество сигналов и слотов;
- сообщения, посылаемые посредством сигналов, могут иметь множество аргументов любого типа;
- сигнал можно соединять с различным количеством слотов. Высылаемый сигнал, в этом случае, поступит ко всем подсоединенным слотам;
- слот может принимать сообщения от многих сигналов, принадлежащих разным объектам;
- соединение сигналов и слотов можно производить в любой точке приложения;

Сигналы и слоты являются механизмами, обеспечивающими связь между объектами. Более того, эта связь может выполняться между объектами, которые находятся в различных потоках.

При уничтожении объекта происходит автоматическое разъединение всех сигнально-слотовых связей. Это гарантирует, что сигналы не будут высылаться к несуществующим объектам.

Нельзя не упомянуть и о недостатках, связанных с применением сигналов и слотов:

- сигналы и слоты не являются частью языка C++, поэтому требуется запуск дополнительного препроцессора перед компиляцией программы;
- отсылка сигналов немного медленнее, чем обычный вызов функции, который производится при использовании механизма функций обратного вызова;
- существует необходимость в наследовании от класса QObject;

В процессе компиляции не производится никаких проверок: имеется ли сигнал или слот в соответствующих классах или нет; совместимы ли сигналы и слот друг с другом и могут ли они быть соединены вместе. Об ошибке можно будет узнать лишь тогда, когда приложение будет запущено. Вся эта информация, выводится на консоль, поэтому, для того чтобы увидеть ее в Windows, в проектном файле необходимо добавить в секции CONFIG опцию console (для Linux никаких дополнительных изменений проектного файла не требуется).

Если есть возможность соединения объектов, то должна существовать и возможность их разъединения. В Qt, при уничтожении объекта, все связанные с ним соединения уничтожаются автоматически, но в редких случаях может возникнуть необходимость в уничтожении этих соединений «вручную». Для этого существует статический метод disconnect(), чьи параметры аналогичны параметрам статического метода connect(). В общем виде этот метод выглядит следующим образом:

QObject::disconnect (sender, signal, receiver, slot);

Существуют два сокращенных, нестатических варианта: disconnect (signal, receiver, slot) и disconnect (receiver, slot).

1.8.3. Виджеты графического интерфейса библиотеки Qt

Виджеты - это объекты классов, наследованных от класса QWidget. QWidget – это класс, использующий множественное наследование. QWidget – это QObject, то есть может иметь родителя, сигналы, слоты и “управляемых” детей. QWidget – это QPaintDevice (базовый класс всех объектов, которые могут быть нарисованы).

Объекты класса QWidget взаимодействуют со своими детьми известными путями. Виджет, не имеющий родителя, называется окном. Если один виджет – родитель другого виджета, то границы виджета-ребенка сливаются с границами родителя.

Предполагается, что QWidget – наименьший из классов GUI, потому что предоставляет пустой четырехугольник. Но сам по себе он достаточно сложен и предоставляет сотни функция. Использование QWidget и его подклассов подобно стоянию на плечах гигантов. Каждый QWidget построен, как наивысший уровень кода библиотеки Qt, которая в свою очередь – вершина различных уровней родной графических библиотек, зависящих от платформы (X11 в Linux, Cocoa в MacOS и Win32 в Windows). QWidget – реализация паттерна Фасад классов виджетов для каждой из родных библиотек.

Ниже (рис.4) представлена иерархия классов графического интерфейса.

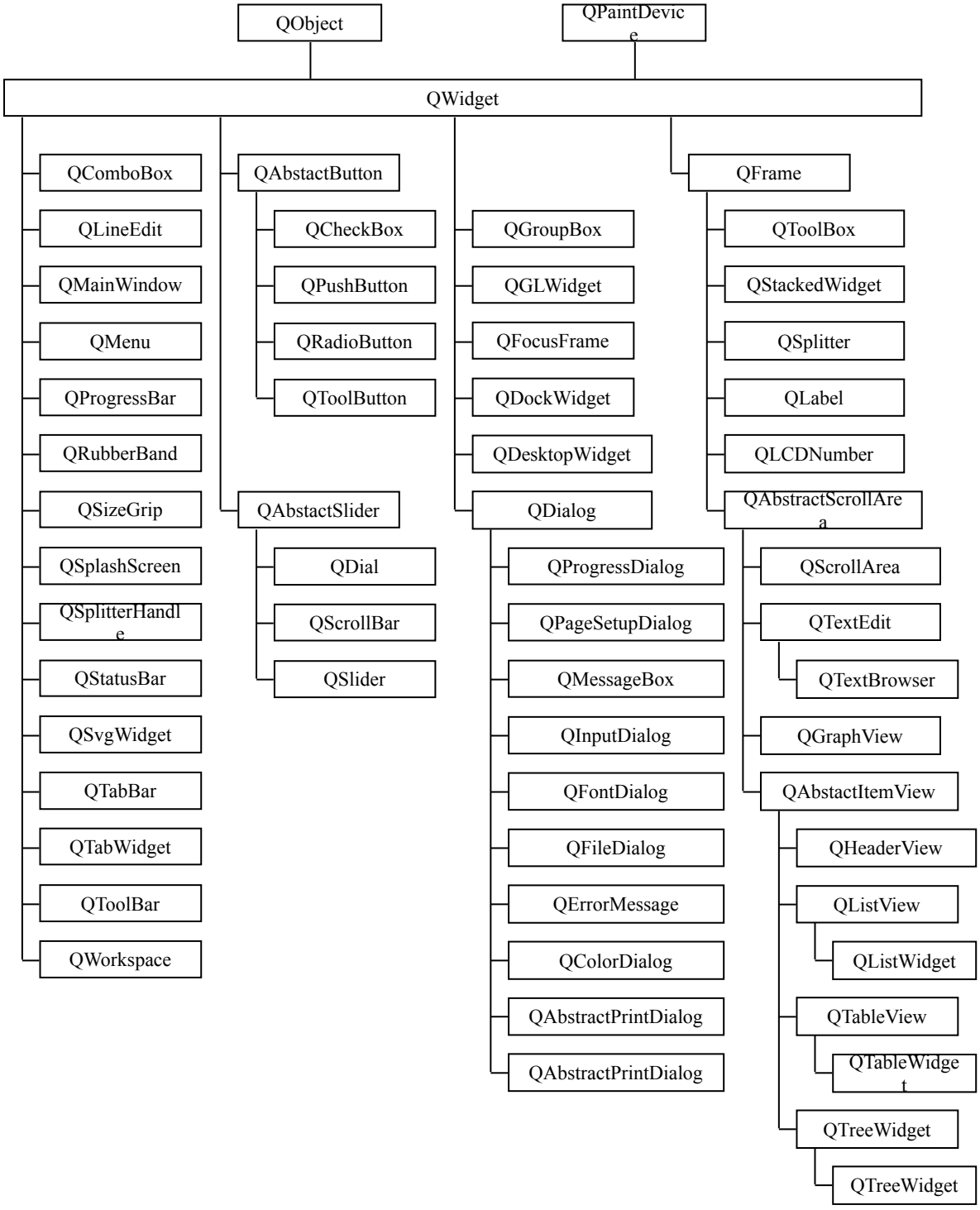

QObject
QPaintDevic
e
QWidget
QComboBox
QLineEdit
QMainWindow
QMenu
QProgressBar
QRubberBand
QSizeGrip
QSplashScreen
QSplitterHandl
e
QStatusBar
QSvgWidget
QTabBar
QTabWidget
QToolBar
QWorkspace
QAbstactButton
QCheckBox
QPushButton
QRadioButton
QToolButton
QAbstactSlider
QDial
QScrollBar
QSlider
QGroupBox
QGLWidget
QFocusFrame
QDockWidget
QDesktopWidget
QDialog
QProgressDialog
QPageSetupDialog
QMessageBox
QInputDialog
QFontDialog
QFileDialog
QErrorMessage
QColorDialog
QAbstractPrintDialog
QAbstractPrintDialog
QFrame
QToolBox
QStackedWidget
QSplitter
QLabel
QLCDNumber
QAbstractScrollAre
a
QScrollArea
QTextEdit
QTextBrowser
QGraphView
QAbstactItemView
QHeaderView
QListView
QListWidget
QTableView
QTableWidge
t
QTreeWidget
QTreeWidget

Рисунок 4.

1.8.4. Классификация виджетов

Виджеты Qt могут быть классифицированы для более легкого поиска нужного класса. Более сложные виджеты могут относиться более чем к одной категории.

Существует четыре категории виджетов.

1. Виджеты кнопок (QPushButton, QToolButton, QCheckBox, QRadioButton).
2. Виджеты ввода (QComboBox, QLineEdit, QTextEdit, QSlider, QTimeEdit, QDateEdit, QDial, QSpinBox, QScrollbar).
3. Виджеты отображения (неинтерактивные виджеты такие как QLabel, QProgressBar и QPixmap).
4. Виджеты контейнеров, такие как QMainWindow, QFrame, QToolBar, QTabWidget и QStackedWidget, включающие другие контейнеры.

Виджеты, представленные выше, используются как строительные блоки для создания других более сложных виджетов, таких как:

- Диалоги для взаимодействия с пользователем (задавать вопросы, предоставлять необходимую информацию), то есть QFileDialog, QInputDialog и QErrorMessage.
- Виджеты для отображения совокупностей данных, таких как QListView, QTreeView и QTableView.

В дополнение к вышеперечисленным классам библиотеки Qt также существуют классы, не имеющие графического представления, но они необходимы во время разработки GUI- интерфейса. Они включают:

- Типы данных (QPoint и QSize) – популярные типы, используемые при работе с графическими объектами.
- Классы, обеспечивающие контроль: QApplication и QAction; оба объекта управляют контролем управления графических приложений.
- Классы разметки. Существует несколько разновидностей: QHBoxLayout, QVBoxLayout, QGridLayout и т.д.
- Классы моделей. QAbstractItemModel и классы, наследованные от него (QAbstractListModel и QAbstractTableModel) являются модель/вид частью фреймворка Qt и используются как базовые классы для классов, которые предоставляют данные для классов QListView, QTreeView или QTableView.
- Модели базы данных. Необходимы для использования с QTableView (или другими классами вида), используя базы данных SQL, как источники данных: QSqlTableModel и QSqlRelationalModel

1.8.5. Менеджеры компоновки (Layout Managers)

Лайауты (Layouts) - это классы размещений/компоновки, и одной из сильных сторон Qt является возможность их использования. Это контейнер, которые после изменения размеров окна автоматически проводят в соответствие размеры и координаты виджетов, находящихся в нем. Они ничего не добавляют к функциональной части самой программы, тем не менее, очень важны для отображения внешнего вида. Лайаут определяет расположение различных виджетов относительно друг друга. Это упрощает разработку приложений, избавляя от недостатков размещения элементов графического интерфейса “вручную”. Классы лайаутов библиотеки Qt берут эту непростую работу на себя. Более того, обладают возможностью инвертирования направления размещения элементов, что может быть полезно пишущим справа налево, например, в Японии. Qt предоставляет так называемые лейаут-менеджеры (менеджеры компоновки), позволяющие организовать размещение виджетов на поверхности другого виджета.

Работа менеджеров компоновки базируется на том, что каждый виджет может сообщить о том, сколько ему необходимо места, может ли он быть растянут по вертикали и/или горизонтали и т.д.

Менеджеры компоновки представляют возможности для горизонтального, вертикального и табличного размещения. Они способны управлять размещением не только виджетов, но и самих лайаутов, что дает возможность создавать довольно сложные размещения.

Класс Layout является основным для всей группы менеджеров компоновки, получился в результате наследования сразу от двух классов: QObject и QLayoutItem. Последний определен в заголовочном файле QLayout. Иерархия классов менеджеров компоновки представлена на рисунке 5.

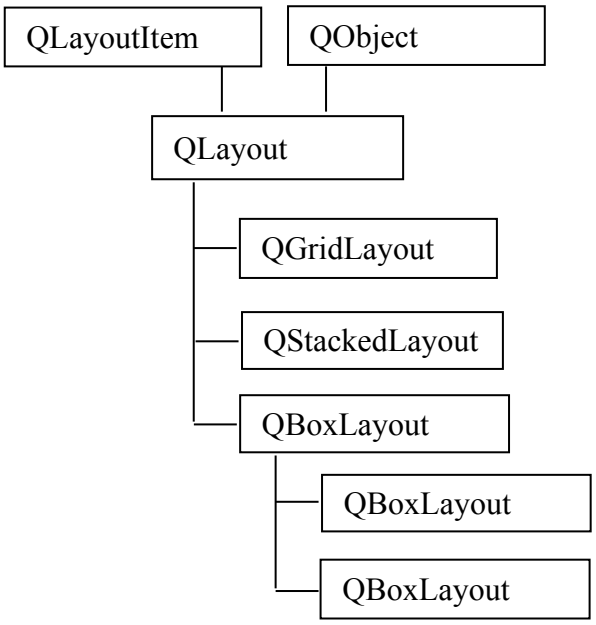


Рисунок 5.

Класс QGridLayout управляет табличными размещениями, а QBoxLayout наследует два класса: QHBoxLayout и QVBoxLayout (для горизонтального и вертикального размещения).

По-умолчанию между виджетами остается небольшое расстояние, необходимое для их визуального разделения. Задать его можно с методом setSpacing(), передав в него нужное значение в пикселях. Методом set Margin() можно установить толщину бордюра лайауты, обычно типичными значениями могут быть 5 или 10 пикселов.

Посредством метода addWidget() производится добавление виджетов в менеджер компонентов, а с помощью метода addLayout() можно добавлять и сами менеджеры компоновки. Если понадобится удалить какой-либо виджет из менеджера компонентов, то следует воспользоваться методом removeWidget(), передав ему указатель на этот виджет.

Табличное размещение QGridLayout

Для табличного размещения используется класс QGridLayout, с помощью которого можно быстро создавать сложные по структуре размещения. Таблица состоит из ячеек, позиции которых задаются строками и столбцами. Добавить виджет в таблицу можно с помощью метода QGridLayout::addWidget(), передав ему позицию ячейки, в которую помещается виджет. Иногда необходимо, чтобы виджет занимал сразу несколько позиций, что достигается с помощью того же метода QGridLayout::addWidget(), в параметрах которого следует указать количество строк и столбцов, которое будет занимать виджет. В последнем параметре можно указывать выравнивание, например, по центру:

playout->addWidget(widget, 17, 1, Qt::AlignCenter);

Фактор растяжения устанавливается методами QGridLayout::setRowSrtetch() и QGridLayout::setColumnStretch(), но не для каждого виджета в отдельности, а для строки или столбца. Расстояние между виджетами также устанавливается для столбцов или строк методом QGridLayout::setSpacing().

Класс QBoxLayout

Для горизонтального или вертикального размещения можно воспользоваться классом QBoxLayout, или унаследованными он него классами QHBoxLayout и QVBoxLayout.

Объект класса QBoxLayout может управлять как горизонтальным, так и вертикальным размещением. Для этого служит первый параметр, который может принимать следующие значения:

- QBoxLayout::LeftToRight (горизонтальное размещение, заполнение производится слева направо);
- QBoxLayout::RightToLeft (горизонтальное размещение, заполнение - справа налево);
- QBoxLayout::TopToBottom (вертикальное размещение, заполнение – сверху вниз);
- QBoxLayout::BottomToTop (вертикальное размещение, заполнение – снизу вверх).

Этот класс расширяет класс QLayout методами вставки на заданную позицию: виджета - QBoxLayout::insertWidget(), менеджера компонентов – QBoxLayout::insertLayout(), расстояния между виджетами – QBoxLayout::insertSpacing() и фактора растяжения - insertStretch(). К менеджеру компонентов с помощью метода QBoxLayout::addSpacing() можно добавить заданное расстояние между виджетами. Класс QBoxLayout определяет свой собственный метод QBoxLayout::addWidget(), для добавления виджетов в менеджер компонентов с возможностью указания фактора растяжения. Фактор растяжения подобен пружине, находящейся между виджетами и имеющей различную упругость, в соответствии с величиной.

Классы QHBoxLayout и QVBoxLayout, унаследованные от QBoxLayout, отличаются от него тем, что в их конструктор не передается параметр, сообщающий о способе размещения – горизонтальный или вертикальный, так как порядок размещения заложен уже в самом классе. Эти классы сами производят горизонтальное или вертикальное размещение, слева направо или сверху вниз.

1.8.6. События

События генерируются оконной системой или Qt в ответ на различные действия. Событие можно охарактеризовать как механизм оповещения приложения о каком-либо происшествии, например, нажатие пользователем кнопки мыши или клавиши клавиатуры, необходимости перерисовки содержимого окна и т.д.

События не следует путать с сигналами. Как правило, сигналы полезны при использовании виджета, в то время как события полезны при реализации виджета.

Класс QEvent инкапсулирует представление о событии. QEvent – это базовый класс для некоторых специализированных классов событий: таких как QActionEvent, QFileOpenEvent, QHoverEvent, QInputEvent, QMouseEvent и др. Объект класса QEvent может быть создан окном в ответ на действия пользователя (т.е. QMouseEvent) в течение некоторого интервала времен (QTimerEvent) или явно из приложения. Функция type() возвращает информацию, с помощью которой можно определить различные виды событий из сотни различных. Также класс QEvent включает, так называемый флаг разрешения события (accept). Флаг может быть установлен вызовом accept() и сброшен – ignore(). В случае если в обработчике события флаг сбрасывается, то получаемые в результате нежелательные событий могут быть переданы родительскому виджету. По-умолчанию, флаг разрешения установлен. В тоже время нет никакой гарантии, что флаг не будет сброшен, например, в конструкторе какого-либо класса, наследованного от данного.

Объекты могут посылать информацию друг другу различными способами. QWidget’ы посылают QEvent’ы другим QObject’ам в ответ на пользовательские действий такие как щелчки мыши и события клавиатуры. Объект класса QWidget также отвечает на события, такие как перерисовка, изменение размера или закрытие. Более того, объекты класса QObject могут передавать информацию друг другу, используя сигналы и слоты.

События уведомляют объекты о себе при помощи своих функций event(), унаследованных от класса QObject(). Реализация QWidget::event() передает большинство обычных событий конкретным обработчикам событий:

- mouseDoubleClickEvent(QMouseEvent *);
- mousePressEvent(QMouseEvent *);
- mouseMoveEvent(QMouseEvent *);
- keyPressEvent(QKeyEvent *);
- keyReleaseEvent(QKeyEvent *);
- paintEvent(QPaintEvent *);
- closeEvent(QCloseEvent *);
- и др.

Всего в Qt предусматривается пять уровней обработки и фильтрации событий:

1. Переопределить конкретный обработчик событий (представляет собой очень распространенный способ обработки событий).
2. Переопределить функцию QObject::event() (в функции event() можно обрабатывать события до того, как они дойдут до обработчиков соответствующий событий. Этот способ используется для обработки редких событий, для который не предусмотрены отдельные обработчики событий. При переопределении функции event() необходимо вызывать функцию базового класса event() для реагирования на необрабатываемые в программе событий).
3. Установить фильтр событий для отдельных объектов QObject (после регистрации объекта с помощью функции installEventFilter() все события целевого объекта сначала предаются функции контролирующего объекта eventFilter(). Если для одного объекта задано несколько фильтров, то они действуют поочередно, начиная с установленного последним).
4. Установить фильтр событий для объекта QApplication (после регистрации фильтра для уникального объекта класса QApplication каждое событие каждого объекта приложения передается функции eventFilter() до его передачи любым другим фильтрам событий, что очень удобно при отладке).
5. Создать подкласс QApplication и переопределить функцию notify() (Qt вызывает QApplication::notify() для генерации события. Переопределение этой функции представляет собой единственный способ получения доступа ко всем событиям до того, как ими займутся фильтры событий).

1.8.7. Drag-and-Drop

Функциональная особенность под названием drag-and-drop обеспечивает перемещение информации с помощью мыши между двумя виджетами как в одном, так и между двумя приложениями. Данная возможность реализуется посредством механизма вызова событий.

Класс QWidget обладает всеми необходимыми методами для поддержки технологии перетаскивания, а некоторые из классов иерархии виджетов уже содержат ее полную реализацию.

Для проведения перетаскивания Qt предоставляет класс QDrag, а для размещения данных различных типов при перетаскивании – класс QMimeData. В этом классе уже реализованы методы для различных типов:

- цветовые значения – setColorData();
- растровые изображения – setImageData();
- текстовая информация – setText();
- гипертекстовая информация в формате HTML – setHtml();
- списки (ссылки) URL (Uniform Resource Locator, единообразный определитель ресурсов) – setUrls(), часто применяемый для перетаскивания файлов.

Не смотря на наличие данных методов, их не хватает, так как может понадобиться перетаскивать и принимать свои собственные типы данных (например, звуковые данные). В этом случае следует использовать метод setData(), в который первым параметром нужно передать строку, характеризующую тип данных, а вторым сами данные в объекте класса QByteArray.

Программирование поддержки перетаскивания можно условно разделить на две части: первый включает в себя реализацию кода для перетаскивания объекта (drag), а вторая – реализует область приема для сбрасываемых в нее объектов (drop). В это время вторая часть должна распознавать, в состоянии ли она принять перетаскиваемый объект или нет.

Реализация Drag

Реализация первой части перетаскивания начинается с перезаписи методов mousePressEvent() и mouseMoveEvent(). В первом из этих методов сохраняется позиция указателя мыши, в котором была нажата кнопка. Эта позиция пригодится в методе mouseMoveEvent() для определения момента старта операции перетаскивания.

Например, как в следующем примере

```
virtual void mousePressEvent (QMouseEvent* pe)
{
    if (pe->button() == Qt::LeftButton) {
        m_ptDragPos = pe->pos();
    }
    QWidget::mousePressEvent(pe);
}

virtual void mouseMoveEvent (QMouseEvent* pe)
{
    If (pe->buttons() & Qt::LeftButton) {
        int distance = (pe->pos() – m_ptDragPos).manhattanLength();
        if (distance > QApplication::startDragDistance()) {
            QMimeData* pMimeData = new QMimeData;
            //сохранение мета-информации
            pMimeData->setText(“This is draggable text”);
            QDrag* pDrag = new QDrag(this);
            pDrag->setMimeData(pMimeData);
            //сохранение данных
            //например
            //pDrag->setPixmap(…)
            pDrag->start(Qt::MoveAction);
        }
    }
}
```

В приведенном выше примере определение класса Drag содержит атрибут m_ptDragPos, предназначенный для сохранения положения курсора мыши в момент нажатия левой кнопки. Инициализация этого атрибута производится в методе mousePressEvent() только в том случае, если была нажата левая кнопка мыши.

Метод mouseMoveEvent() нужен для распознавания начала перетаскивания. Это сделано потому, что нажатие левой кнопки мыши и последующее перемещение указателя не всегда говорит о желании пользователя перетащить объект, так как у пользователя могла просто дрогнуть рука, случайно переместив указатель мыши. Чтобы быть уверенным, необходимо вычислить расстояние между текущей позицией и той позицией, в которой была нажата левая кнопка мыши. Если это расстояние превышает величину,

возвращаемую статическим методом startDragDistance() (по-умолчанию, 4 пикселя), то тогда можно исходить из того, что перемещение указателя мыши было неслучайным и пользователь действительно хочет перетащить выбранный объект.

Далее создается объект класса QMimeData, в который вызовом метода setText() передается перетаскиваемый текст. Затем создается объект перетаскивания класса QDrag, в конструктор которого передается указатель на виджет, осуществляющий перетаскивание. Метод start() запускает операцию перетаскивания.

Реализация Drop

Рассмотрим работы механизма drop на примере. Создадим виджет, способный принимать и обрабатывать сбрасываемые в него объекты. После сбрасывания виджет отображает полное имя файла.

Файл Drop.h:

```
#ifdef _Drop_h_
#define _Drop_h_

#include <QtGui>

// ============================================================================
Class Drop : public QLabel {
        Q_OBJECT

protected:
        virtual void dragEnterEvent(QDragEnterEvent* pe)
        {
                if(pe->mimeData()->hasFormat("text/url-list")) {
                        pe->acceptProposeAction();
                }
        }

        virtual void dropEvent(QDropEvent* pe)
        {
                QList<QUrl> urlList = pe->mimeData()->urls();
                QString str;
                foreach (QUrl url, urlList) {
                        str += url.toString() + "\n";
                }
                setText("Dropped:\n" + str);
        }
public:
        Drop(QWidget* pwgt = 0) : QLabel("Drop Area", pwgt)
        {
                setAcceptDrops(true);
        }
};
#endif //_Drop_h_
```

Для того чтобы виджет был в состоянии принимать сбрасываемые объекты, в конструкторе класса Drop производится вызов метода setAcceptDrops(), в который передается true. В этом классе для получения сбрасываемых объектов необходимо перезаписать метода dragEnterEvent() и dropEvent().

Метод dragEnterEvent() вызывается каждый раз, когда перетаскиваемые объекты пересекают границу виджета. Этот метод нужен для того, чтобы виджет был в состоянии сообщать о готовности принимать перетаскиваемые объекты, при этом указатель мыши из перечеркнутого круга превращается в стрелку с прямоугольником. В случае если виджет не готов, то внешний вид указателя остается без изменений.

Обычно виджет способен принимать данные всех типов, поэтому необходимо проверять на совместимость тип данных перетаскиваемых объектов с помощью метода hasFormat().

Вызов метода acceptProposedAction() из объекта события сообщает о готовности виджета допустить проводимое действие и принять перетаскиваемый объект.

Класс QDragEnterEvent унаследован от класса QDragMoveEvent. Этот класс предоставляет методы accept() и ignore(), которыми можно воспользоваться для разрешения (или запрещения) принятия перетаскиваемых объектов. В эти методы можно передавать объекты класса QRect. С их помощью можно ограничить размеры принимающей области виджета. Чтобы разрешить, например, сбрасывание объектов во всей области виджета, можно сделать следующий вызов: accept(rest()).

Метод dropEvent() вызывается при сбрасывании перетаскиваемых объектов в пределах окна виджета, что происходит в момент отпускания правой кнопки мыши. Вызов метода urls() из объекта класса QMimeData возвращает список файлов в переменную urlList. Далее, в цикле foreach вызовом метода QUrl::toString() возвращаются строки, которые объединяются в одну, разделяя их символом возврата каретки (\n). После этого полученная строка отображается с помощью метода setText().

Создание собственных типов перетаскивания

Возможности Qt не ограничены перетаскиванием данных определенных типов, таких как, например, текст или растровые изображения. Перетаскиваться может информация любого типа. Для этого необходимо определиться с идентификацией типа для перетаскиваемых данных, для того, чтобы принимающая сторона могла принять решение – допускает она их или нет. Идентификация достигается путем задания имени своего типа в первом параметре метода QMimeData::setData(). Так например,

```
QByteArray itemData;
QDataStream dataStream(&itemData, QIODevice::WriteOnly);
QRect square = …;
QPoint location = …;
QPixmap Pixmap = …;
dataStream << pixmap << location << square;
QMimeData *mimeData = new QMimeData;
mimeData->setData("image/x-puzzle-piece", itemData);
QDrag *drag = new QDrag(this);
drag->setMimeData(mimeData);
drag->setHotSpot(event->pos() - square.topLeft());
drag->setPixmap(pixmap);
if (drag->start(Qt::MoveAction) == 0) {
      //выполняется, когда перетаскивание завершено
}
```

Метод QDrag::setPixmap() задает изображение, отображаемое при перетаскивании. В метод QDrag::HotSpot() передается позиция, на которой будет стоять указатель мыши при перетаскивании. Таким образом, основным отличием программного кода при использовании собственным типов является использование метода QDrag::setData(), первым параметром которого является QString с указанием названия нового типа, а вторым – QByteArray с передаваемыми данными.

1.8.8. Контекстно-независимые представления

Контекстно-независимое представление не зависит от возможностей графической карты вашего компьютера. Данные растрового изображения помещаются в обычный массив, что дает возможность эффективного обращения к каждому из пикселей в отдельности, а также позволяет эффективно производить операции записи и считывания файлов растровых изображений.

QImage является основным классом для контекстно-независимого представления растровых изображений. Класс унаследован от класса контекста рисования QPaintDevice, что позволяет также использовать методы рисования, определенные в классе QPainter.

Класс QImage представляет поддержку для форматов, указанных в табл. 1.

Таблица 1. Перечисление Format класса QImage.

Константа	Описание
Format_Invalid	Растровое изображение недействительно
Format_Mono	Каждый пиксель представлен одним битом. Биты укомплектованы в байте таким образом, что первый является старшим разрядом.
Format_MonoLSB	Каждый пиксель представлен одним битом. Биты укомплектованы в байте таким образом, что первый является младшим разрядом.
Format_Index8	Данные растрового изображения представляют собой 8-битные индексы цветовой палитры.
Format_RGB32	Каждый пиксель представлен 32 битами. В них значение для альфа-канала всегда равно значению 0xFF, то есть непрозрачно.
Format_ARGB32	Каждый пиксель представлен 32 битами.
Format ARGB32_Premultiplied	Практически идентичен формату Format_ARGB32, но код оптимизирован для использования QImage в качестве контекста рисования.

Значение формата можно узнать с помощью метода format(). Для конвертирования растрового изображения из одного формата в другой предусмотрен метод convertToFormat(), который возвращает новый объект QImage. Данные растрового изображения хранятся в объектах класса QImage.

Объект класса QImage можно создать различными способами:

- Передать в конструктор ширину, высоту и формат растрового изображения. Например, строка

  QImage img (320, 240, QImage::Format_RGB32);

- Создает растровое изображение шириной 320 пикселов, высотой 200 и глубиной цвета 32 бит.
- Передать в конструктор имя файла и таким образом считать данные из файла, хранящего растровое изображение. Если данный графический формат поддерживается Qt, то данные будут загружены и автоматически установится высота, ширина и формат.

  QImage img(“lisa.jpg”);

- Использовать метод load() в качестве альтернативы для считывания. С его помощью можно загружать растровое изображение в любой момент. Первый параметром передается имя файла, а вторым – формат (строка типа unsigned char*, принимающее следующее возможное строковое значение: GIF, BMP, JPG, XPM, XBM или PNG. Если во втором параметре вообще ничего не передавать, то класс QImage произведет попытке распознать графический формат самостоятельно.

  QImage img;
  img.load(“lisa.jpg”);

При помощи метода save() можно сохранять в файле растровое изображение из объекта класса QImage. Первым параметром передается имя файла, а вторым – формат, в котором он должен быть сохранен. Например:

```
QImage img (320, 240, 32, QColor::blue);
img.save (“blue.jpg”, “JPG”);
```

Полный перечень поддерживаемых форматов файлов доступен через QImageReader::supportedImageFormats() и QImageWriter::supportedImageFormats(). Дополнительно может быть добавлена поддержка других форматов файлов в виде плагинов.

По умолчанию, Qt поддерживает форматы, представленные в табл. 2.

Таблица 2.

Формат	**Расшифровка**	**Поддержка в Qt**
BMP	Windows Bitmap	Чтение/Запись
GIF	Graphic Interchange Format (опционально)	Чтение
JPG	Joint Photographic Experts Group	Чтение/Запись
JPEG	Joint Photographic Experts Group	Чтение/Запись
PNG	Portable Network Graphics	Чтение/Запись
PBM	Portable Bitmap	Чтение
PGM	Portable Gray Map	Чтение
PPM	Portable Pixmap	Чтение/Запись
TIFF	Tagged Image File Format	Чтение/Запись
XBM	X11 Bitmap	Чтение/Запись
XPM	X11 Pixmap	Чтение/Запись

Объект класса QImage может быть отображен в контексте рисования при помощи метода QPainter::drawImage(). Перед тем как отобразить объект QImage на экране, метод drawImage() преобразует его в контекстно-зависимое представление (объект класса QPixmap). Если нужно вывести только часть растрового изображения, то необходимо указать эту часть в дополнительных параметрах метода drawImage(). Следующий пример иллюстрирует отображение участка растрового изображения, задаваемого координатой (30, 30) и имеющего ширину равную 110, а высоту -100 пикселей.

```
QPainter painter(this);
QImage img(“lisa.jpg”);
painter.drawImage(0, 0, img, 30, 30, 110, 100);
```

В случае вывода всего изображения последняя строка будет выглядеть следующим образом:

```
painter.drawImage(0, 0, img);
```

При помощи метода scaled() можно получить новое растровое изображение с измененными размерами. Сначала задаются размеры нового изображения. Затем следуют параметр, определяющий каким образом изображение будет масштабироваться. Задается флагами:

- Qt::IgnoreAspectRatio (масштабирование до выбранного размера, не учитывая соотношения размеров изображения);
- Qt::KeepAspectRatio (масштабирование с учетом соотношения сторон изображения);
- Qt::KeepAspectRatioByExpanding (масштабирование на весь прямоугольник с минимальным выходом за пределы, сохраняя соотношение сторон).

1.8.9. Контекстно-зависимое представление

Контекстно-зависимое представление позволяет отображать растровые изображения на экране гораздо быстрее, чем контекстно-независимое, так как оно не требует проведения дополнительных преобразований. К объектам контекстно-зависимого представления можно применить все методы класса QPainter. Большинство процессоров графических карт обладают способностью отображать графические примитив, например, линии, полигоны и поверхности, не используя при этом основного процессора, благодаря чему очень сильно ускоряется графический вывод и работа самой программы.

Класс QPixmap унаследован от класса контекста рисования QPaintDevice. Его определение находится в заголовочном файле QPixmap. Объекты класса содержат растровые изображения, не отображая на экране. При необходимости его можно использовать в качестве промежуточного буфера для рисования, то есть если требуется нарисовать изображение, то его можно сначала нарисовать в объекте класса QPixmap, а уже потом скопировать в видимую область.

Для создания объекта этого класса в его конструктор нужно передать ширину и высоту. Например:

```
QPixmap pix (320, 240);
```

Глубина цвета в создаваемом объекте класса QPixmap автоматически будет установлена в соответствии с актуальным значением графического режима, которое доступно с помощью метода QPixmap::defaultDepth().

Еще один способ создания объекта данного класса – передача имя файла растрового изображения прямо в конструктор.

```
QPixmap pix (“forest.jpg”);
```

Так как объекты класса QPixmap содержат не сами данные, а их идентификаторы, поэтому прямой доступ к каждому пикселю в отдельности очень медленный. В связи с чем разумнее будет воспользоваться классом QImage.

Для объектов класса QPixmap доступны методы load() и save() для сохранения и записи графических изображений. При проведении этих операций будет осуществляться промежуточное конвертирование из объекта класса QImage (или в него).

Для отображения объекта класса QPixmap в видимой области используется метод QPainter::drawPixmap(). Ниже приведен пример с указанием двух различных вариантов вызова метода drawPixmap().

```
QPainter painter(this);
QPixmap pix(“forest.jpg”);
…
//так
painter.drawPixmap(0, 0, pix);
// или так
QRect r(pix.width(), 0, pix.width() / 2, pix.height());
painter.drawPixmap(r, pix);
```

1.8.10. Встроенные диалоги в Qt

Многие диалоги уже созданы предварительно. Один из таких диалогов – диалог открытия файла, реализованный в QFileDialog,

Диалоги с сообщениями

Очень часто программа должна предоставлять информацию пользователю.

В качестве решения этой задачи есть возможность использовать следующий способ. Наследовать отдельный класс от QDialog, который бы отображал сообщения и дополнительно предоставлял одну или несколько кнопок для выбора дальнейших действий. Данная задача является очень распространенной, и поэтому в Qt уже реализован класс QMessageBox для этих целей. QMessageBox может включать до 3 кнопок и содержать иконку для сообщения. Текст диалога может быть как и стандартным текстом, так и размеченным с использованием HTML-тэгов.

После определения заголовка и сообщения, определяемых в первых двух параметрах, существует возможность выбора одного из четырех предопределенных иконок путем отправления одного из следующих значений в качестве третьего аргумента:

Таблица 3.

QMessageBox::Question	Предназначен для вопросов.
QMessageBox::Information	Подчеркивает важность информации.
QMessageBox::Warning	Следует использовать для потенциально опасных действий.
QMessageBox::Critical	Предпочтительно для выделения серьезных ошибок.
QMessageBox::NoIcon	Не отображает иконок вообще.

Если же стандартный набор иконок не удовлетворяет необходимых потребностей, то можно вместо этого заменить любым QPixmap, который вы хотите использовать, как иконку для окна с сообщением, используя метод setPixmapIcon(). Параметры в конструкторе с четвертого по шестой используются для того, чтобы задать до четырех различных кнопок. Возможные значения для этих параметров приведены ниже.

Таблица 4.

QMessageBox::Ok	Ok
QMessageBox::Cancel	Отмена
QMessageBox::Yes	Да
QMessageBox::No	Нет
QMessageBox::Abort	Прервать
QMessageBox::Retry	Повтор
QMessageBox::Ignore	Игнорировать
QMessageBox::YesAll	Да, все
QMessageBox::NoAll	Нет, все
QMessageBox::NoButton	–

Через операцию “ИЛИ” с QMessageBox::Default и QMessageBox::Escape, можно также задать действия, привязанные к кнопкам Enter или Esc.

```
QString text = QObject::tr(“<qt> This is a <b>very</b> complicated way”
                    “of showing message boxes. <i> Only use this in exceptional cases</i>!”
                    “Do you want to continue?</qt>”);
QMessageBox msg(QObject::tr(“Academic Example Warning”), text,
                    QMessageBox::Warning, QMessageBox::Yes | QMessageBox::Default,
                    QMessageBox::No | QMessageBox::Escape, QMessageBox::NoButton);
if (msg.exec() == QMessageBox::Yes)
{
    qDebug() << “Keep on going!”;
}
```

Сообщения об ошибках

Кроме QStatusBar::showMessage() и QMessageBox существует третья возможность сообщить пользователю информацию, используя класс QErrorMessage.

В отличие от QMessageBox, QErrorMessage не предоставляет статистических методов. Он только отображает сообщения в случае вызова слота showMessage().

Существуют две особенности, делающие этот класс очень используемым. Во-первых, если два сообщения ждут, как в следующем примере, тогда только QErrorMessage показывает следующее сообщение после того, как закрыто первое. Во-вторых, каждое из этих диалогов включают “Показать это сообщение снова”. Если пользователь сбрасывает его, Qt показывает сообщение об ошибке, если эта же ситуация возникнет в будущем.

Диалоги выбора файла

Приложения часто используют диалоги для запроса пользователя о выборе файлов или каталогов. По этой причине почти все платформы обеспечивают реализацию каждого из этих диалогов.

Класс QFileDialog, который реализует выбор файла или каталога, можно использовать двумя способами:

- Если создавать класс с использованием конструктора, Qt отобразит отдельный диалог.
- Если использовать предопределенный статический метод, часто используемый программистами, тогда Qt попробует использовать диалог выбора файла, свойственной каждой отдельной операционной системе.

Если существует потребность не использовать диалоги системы, следует вызывать статические методы с флагом QFileDialog::DontUseNativeDialog. В этом случае Qt использует свой диалог. Этот и другие флаги задокументированы в таблице 5.

Таблица 5.

Значение	Результат
QFileDialog::ShowDirsOnly	Показывает только папки
QFileDialog::DontResolveSymlinks	Не следовать символическим ссылкам, но интерпретировать их как файлы или папки.
QFileDialog::DontConfirmOverwrite	Не подтверждать, если существующий файл должен быть перезаписан
QFileDialog::DontUseSheet	Не отображать диалог выбора файла в Mac OS X как страницу. Работает если флаг DontUseNativeDialog сброшен.
QFileDialog::DontUseNativeDialog	Всегда использовать диалог библиотеки Qt

Рассмотрим пример.

```
QString file = QFileDialog::getOpenFileName(this,
                tr(“Pick a File”),
                “/home”,
                tr(“Images (*.png *.xpm *.jpg)”));
```

В случае успеха, этот метод возвращает имя файла, в противном случае, возвращает нулевую строку QString (которая может быть определена с помощью file.isNull()). В качестве первого аргумента метод ожидает указатель на родительский виджет диалога. Если же установить первый аргумент как нулевой указатель, то диалог будет не модальным. Следующие два аргумента являются заголовком и стартовой папкой.

Класс QDir предоставляет статические методы для получения путей для большинства важных папок, которые могут быть использованы как третий аргумент вместо фиксированной строки:

QDir::currentPath()

Возвращает текущую папку приложения.

QDir::homePath()

В Windows, возвращает содержимое переменной HOME. Если же она не существует, то возвращает переменную USERPROFILE. Если же и в этом случае возникают ошибки, то Qt возвращает папку из HOMEDRIVE и HOMEPATH. Если же и эти переменные не установлены, метод вызывает rootPath(). В системе Unix, включая OS X, метод использует переменную HOME. Если же и она не установлена, используется rootPath().

QDir::rootPath()

В Windows возвращает “C:\”, в Unix корневой каталог “/”.

QDir::tempPath()

Возвращает /tmp в Unix, или значение, основанное на переменных TEMP и TMP, в Windows.

Наконец, в качестве четвертого аргумента, getOpenFileName() ожидает фильтр файлов. Это позволяет определять только файлы со специальным расширением. Здесь следует придерживаться следующего синтаксиса:

“filetypedesignator (*.ex1 *.ex2 … *.exn)”

Filetypedesignator показывает пользователю какой тип файлов отображается. Расширения файлов, представленных далее действует как фильтр для имен файлов, включаемых в папку. Следующий фильтр, например, найдет все файлы с расширением .png, .xpm или .jpg:

“Изображения (*.png *.xpm *.jpg)”

Есть также возможность задавать несколько фильтров. Для этого следует разделять двумя точками с запятой, как в следующем примере.

“Image (*.png *.xpm *.jpg);;Text files (*.txt)”

В общем случае, количество фильтров не ограничено.

В случае, когда необходимо выбрать несколько файлов, следует использовать статическим методом getOpenFileNames(). В сравнении с getOpenFileName() этом метод возвращает QStringList, но параметры метода при этом остаются теми же. Каждая запись в возвращенном списке соответствует выбранному файлу.

Следующий пример выводит все пути к выбранным файлам.

```
QStringList fileList = QFileDialog::getOpenFileName (
                this,
                tr(“Pick a File”),
                “/home”,
                tr(“Images (*.png *.xpm *.jpg)”));
foreach (QString file, fileList)
                qDebug() << file;
```

Для того чтобы пользователь имел возможность выбора только среди папок, используется метод getExistingDirectory(). Он не содержит фильтра и отображает только папки. Более того, также проверяется существование папки.

В случае, когда пользователь хочет сохранить файл, обычно последний еще не существует. getOpenFileName() проверяет существование заданного файла, так что его нельзя использовать для сохранения файлов. Поэтому в Qt предусмотрен другой статический метод getSaveFileName(). По параметрам и возвращаемому значению он аналогичен методу getOpenFileName().

Диалоги ввода

Для простых запросов Qt предоставляет различные шаблоны диалогов ввода в зависимости от ситуаций, содержащие виджет ввода и две кнопки: OK и Cancel.

Для ввода целочисленных значений используется метод класса QInputDialog getInteger().

```
bool ok;
int alter = QInputDialog::getInteger (this, tr("Enter Age"),
      tr("Please enter year of birth"), 1982,
      1850, QDate::currentDate().year(), 1, &ok);
if (ok)
{
      …
}
```

Первый аргумент – ссылка на родительский виджет. Второй – текст заголовка, третий – пояснительный текст, отображаемый внутри виджета. Далее следуют 2 параметра – пределы изменения вводимого значения. Затем идет параметры с текущим значением, шагом изменения величины в виджете. И наконец, последний параметр, ссылка на переменную типа bool, сообщает пользователю была ли нажата левая или правая кнопка в диалоге ввода. Так если пользователь нажал кнопку Отмена, будет возвращено значение false, в противном случае возвращается true.

Для ввода чисел с плавающей точкой используется статический метод QInputDialog::getDouble(). Его параметры аналогичны параметрам метода getInteger() за исключением предпоследнего, где теперь ожидается число цифр числа после точки.

Метод QInputDialog::getItem() позволяет пользователю выбирать строку из нескольких предопределенных значений.

```
QStringList languages;
bool ok;
Languages << "English" << "German" << "French" << "Spanish";
QString language = QInputDialog::getItem(this, tr("Select Language"),
                        tr("Please select your language"), languages,
                        0, false, &ok);
if (ok) {
      …
}
```

Для чтения текста используется метод QInputDialog::getText().

```
QString passd = QInputDialog::getText(this, tr("Please Enter Password"),
                        tr("Please enter a password for '%1'").arg(resource),
                        QLineEdit::Password, QString(), 0);
```

Первые три параметра задают соответственно родительский виджет, заголовок диалога и текст диалога. Далее следует параметр, задающий способ отображения в поле ввода, он может принимать значения:

- QLineEdit::NormalMode. Текст отображается, как вводится
- QLineEdit::NoEcho. Не печатает ничего вообще.
- QLineEdit::Password. Отображает символ-замену для печати каждого введенного символа. Обычно используется звездочка или иконка с кружком.

Предпоследним параметром передается строка по-умолчанию. Последний параметр в этом примере – 0 (т.е. нулевой указатель) вместо указателя на булеву переменную, т.к. в этом случае можно проверить полученное значение с помощью функции QString::isEmpty() для определения введенного значения.

1.9. Модуль QtNetwork

Библиотека QtNetwork предоставляет классы для написания сетевых приложений. В дополнение к наиболее простым классам коммуникационных сокетов QTcpSocket и QUdpSocket эта библиотека также содержит классы для клиентских программ для работы с HTTP и FTP (QHttp и QFtp соответственно).

В отличие от QtGui, QtNetwork требует модуль QtCore, но может также использоваться совместно с модулем QtGui и другими библиотеками.

1.9.1. Создание клиент-серверных приложений, используя Qt

Программы-сервера обычно не требуют графического пользовательского интерфейса и имеют тенденцию быть запущенными в фоновом режиме невидимо для пользователя. То есть возможность создавать приложения, не включая модуль графического пользовательского интерфейса. Следует учитывать две особенности:

- QApplication заменяется на QCoreApplication;
- Необходимо добавить строку QT -= gui в файл проекта.

В результате получается приложение никоем образом не связанное с графическим интерфейсом, предоставляемым Qt, так что требуется меньше дискового пространства и необходимо меньше памяти, как во время запуска приложения, так и при распространении.

Классы QTcpSocket и QTcpServer могут использоваться для реализации клиентов и серверов TCP. TCP – это транспортный протокол, который составляет основу большинства прикладных протоколов сети Интернет, включая FTP и HTTP, и который может также использоваться для создания пользовательский протоколов. TCP является потоориентированным протоколом. Для приложений данные представляются в виде большого потока данных, очень напоминающего большой однородный файл. Протоколы высокого уровня, построенные на основе TCP, являются либо строкоориентированными, либо блокоориентированными:

- строкоориентированне протоколы передают текстовые данные построчно, завершая каждую строку символом перехода на новую строку;
- блокоориентированные протоколы передают данные в виде двоичных блоков. Каждый блок имеет в начала поле, где указан его размер, и затем идут байты данных.

Класс QTcpSocket наследует QIODevice через класс QAbstractSocket, и поэтому чтение с него или запись на него могут производится с применением средств класса QDataStream или QTextStream. Одно существенное отличие чтения данных из сети по сравнению с чтением обычного файла заключается в том, что необходимо быть уверенным в получении достаточного количества данных от партнерского узла перед использованием оператор >>.

Приложение, созданное мной, использует пользовательский протокол блочной передачи.

1.9.2. Класс QTcpSocket

При работе объект класса QTcpSocket сообщает о своем текущем состоянии путем вызовов сигналов. В моем приложении-клиенте, исходный текст которого содержится в приложении 2, отлавливаются сигналы readyRead(), error(), connected() и disconnected(). Класс QTcpSocket наследует от класса QAbstractSocket шесть сигналов, добавленные в этот класс после наследования от QIODevice. Наиболее важными из них являются:

- connected() (возникает, когда вызов connectToHost() успешно выполнен и соединение с сервером установлено);
- disconnected() (возникает при отсоединении от сервера);
- error(QAbstractSocket::SocketError) (вызывается в случае возникновения ошибки, аргумент сообщает причину);
- hostFound() (вызывается, когда вызов connectToHost() успешно завершен и имя хоста найдено и определен его адрес; этот сигнал отсылается до реального соединения, в следствии чего нет гарантии успешного установления соединения, и сервер еще может отказаться принимать его);
- stateChanged(QAbstractSocket::SocketState) (возникает при изменении состояния соединения);
- readyRead() (вызывается, когда данные доступны для чтения, и если поступила новая информация, т.е. при невозможности прочитать данные, новый сигнал не вызовется).

Помимо обозначенного выше способа чтения/записи из/в сокет существует метод QIODevice::read()/QIODevice::write(). Метод QIODevice::bytesAvailable() возвращает количество доступных для чтения байт. Для установления соединения с сервером следует вызывать QAbstractSocket::connectToHost().

1.9.3. Класс QTcpServer

Класс QTcpServer делает возможным принимать входящие TCP-соединения. Вызов метода listen() ставит на прослушивание определенный порт. Далее существует два способа обработки входящий соединений:

- Реагирование на сигнал newConnection(), возникающий при соединении с сервером. Далее вызовом метода nextPendingConnection() выбрать ожидающее подключение и получить указатель на объект класса QTcpSocket, который можно использовать для обмена данными с клиентом.

```
void Server::sendData() {
    QByteArray block;
    …
    QTcpSocket *clientConnection = tcpServer->nextPendingConnection();
    connect(clientConnection, SIGNAL(disconnected(),
            clientConnection, SLOT(deleteLater()));
    clientConnection->write(block);
    clientConnection->disconnectFromHost();
}
```

- Создать свой класс путем наследования от QTcpServer и реализовать заново виртуальную функцию incomingConnection(), а с помощью передаваемого в нее дескриптора методом setSocketDescriptor() получить сокет для работы с клиентом.

```
void Server::incomingConnection(int descriptor)
{
     ClientSocket *socket = new ClientSocket(this);
     socket->setSocketDescriptor(descriptor);
}
```

Глава 2. Практическая часть

2.1. Описание игры

Существует огромное количество головоломок. Одной из самых популярных является, так называемый, пазл (Puzzle).

Общий вид приложение n-Puzzle сразу после запуска приведен ниже.

Доступны два режима игры: “Классический” и “n-Puzzle”. По-умолчанию, отмечен режим “n-Puzzle”, но может быть изменен в группе переключателей “Тип игры”.

Игра при режиме “Классический” напоминает собирание настольного пазла, когда картинка собирается из отдельных частей. При этом расположение отдельных кусков изображения можно простым перетаскиванием. Этот режим представляет собой некое подобие самого обычного пазла, когда присутствуют только отдельные части изображения. В то время как цель игры – собрать итоговую картинку из имеющихся частей, верно расположив их в правильном порядке.

Также широкое распространение другой вид пазла, получивший название 15-Puzzle. Состоит из 15 скользящих плиток, каждая (условно) с номером от 1 до15 на каждой. Все 15 плиток собраны в оправу 4x4 с одним пропущенной плиткой. Назовем пропущенную плитку X. Плитка X выбирается произвольным образом.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	X

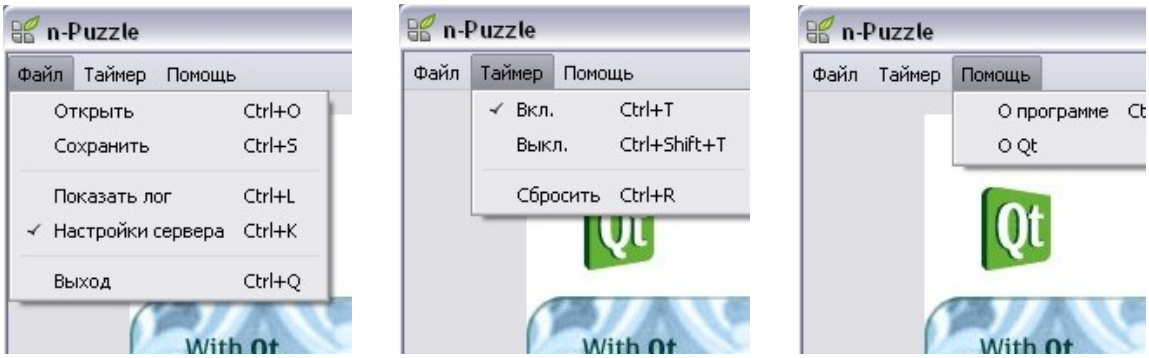
Цель головоломки – привести в порядок плитки так, чтобы они образовывали верную последовательность. Единственной разрешенной операцией является обмен X с одной из близлежащих плиток. Другими словами, плитка смежная с пропущенной (пустой) плиткой, может быть поставлена на место пустой ячейки. Первоначальная настройка стола находится в перемешанном состоянии. Головоломка считается решенной, если достигнуто следующее состояние: пустая плитка находится в нижнем правом углу решетки. Стоит упомянуть, что некоторые состояния головоломки не могут быть разрешены.

В данной реализации головоломки будут использованы кусочки картинок вместо простых ячеек с номерами на них. Цель игры расположить кусочки головоломки в их правильном порядке. Также присутствует возможность выбора количество частей, на которое разбивается изображение. Сами изображения хранятся на отдельном сервере. Доступ к ним обеспечивается через сеть.

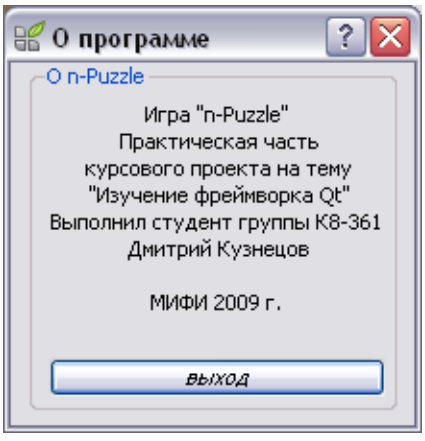
2.2. Клиент

Главное меню.

Всего 3 субменю, созданные на главной панели: “Файл”, “Таймер” и “Помощь”. Для работы приложения в меню “Файл” доступны пункты “Открыть” и “Сохранить” для работы с локальными файлами изображений, пункт “Показать лог” для отображения окна лога внизу окна, пункт “Настройка сервера” для отображения группы элементов управления для настройки подключения к серверу и пункт “Выход” для завершения работы приложения. Субменю “Таймер” включает пункты по принципу переключателя для описания состояние таймера “Вкл.” или “Выкл.”, и еще один пункт – “Сбросить” (устанавливает таймер в 0).

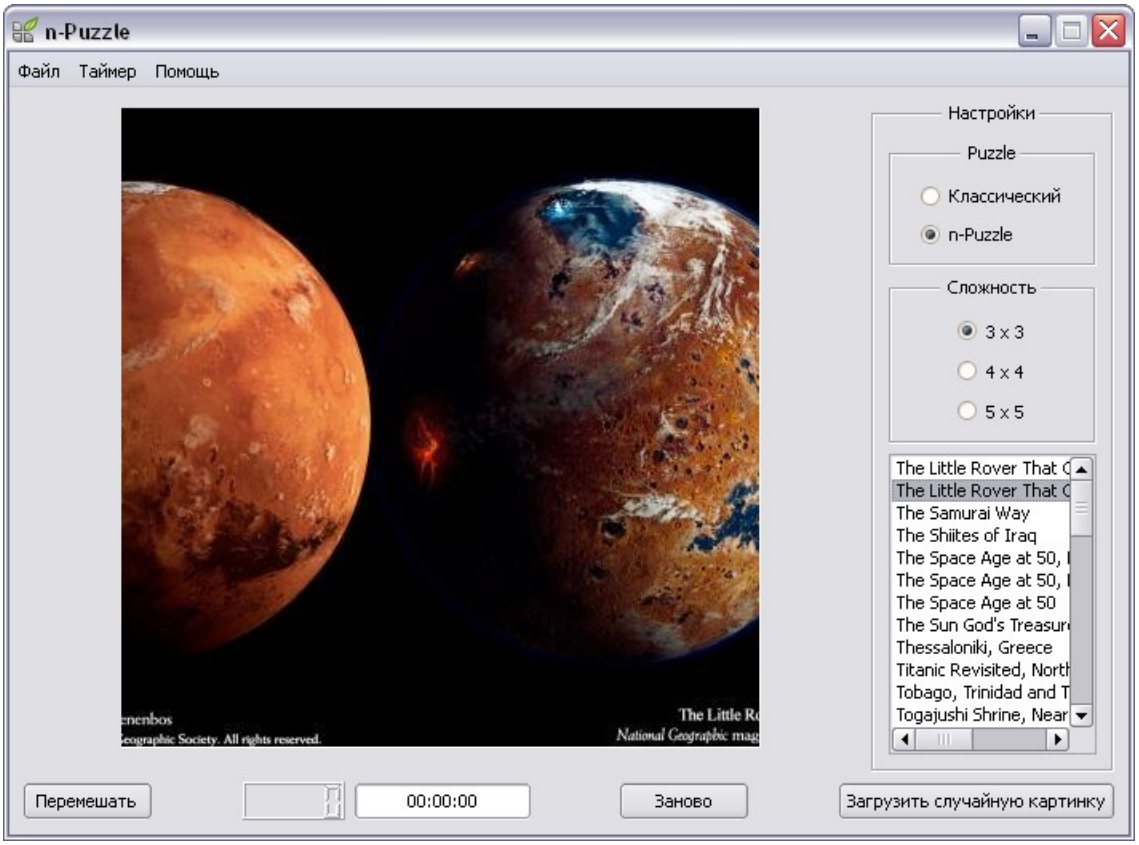


И, наконец, субменю Помощь включает пункт “О программе”. При выборе пункта “О программе” появляется диалог с информацией об игре.



Область головоломки.

Область головоломки представляет собой виджет, разделенный на 9, 16, 25 (в зависимости от настроек игры) участка. Все части объединены в решетку (соответственно, сетка из 3x3, 4x4, 5x5 участков), каждый из которых представляет собой кусочек картинки. В определенное время все за исключением одного участка показывают кусочки изображения. Когда пользователь щелкает по участку, соседнему с пустым, оба участка меняются местами. Предполагается, что картинка будет разбита на равные части. Отдельные изображения получаются путем деления на равные части исходной картинки.



Загрузка картинок.

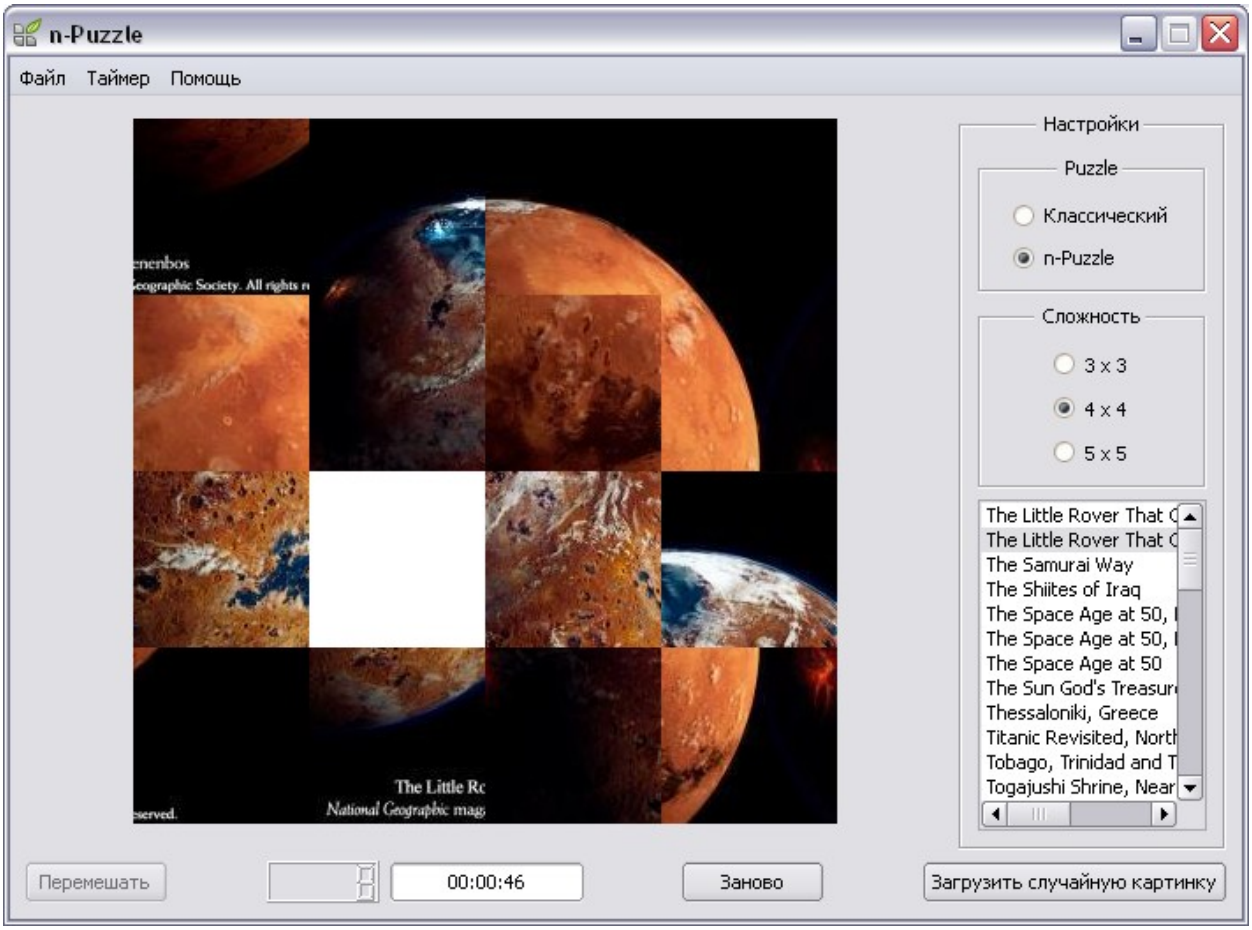
Картинки находятся на сервере. Перемешивание происходит по нажатию на кнопку “Перемешать”, после этого случайным образом выбирается участком, который в последствии окажется пустым и итоговом изображении. Картинки могут быть произвольного размера и иметь любой формат, поддерживающий операцию чтения в библиотеке Qt.

Список изображений.

Список имен файлов, содержащихся в папке на сервере и доступных для загрузки. В случае если сервер недоступен, выдается список кэшированных локальных изображений, загруженных на локальный компьютер при первом обращении к картинке. После того как имя выбрано, программа проверяет наличие файла с таким именем в кэше. В случае отрицательного результата, соединятся с сервером, загружает изображение и заменяет текущую картинку. Кэш также может использоваться как локальное хранилище картинок для приложения в случае отсутствия подключения к серверу. В папку кеша можно поместить любой файл. Если этот файл – картинка, формат которой поддерживается Qt, то это изображение будет использоваться приложением, в противном случае – загрузиться стандартное изображение, которое можно видеть сразу после старта приложения.

Кнопки.

Всего присутствует 4 кнопки на главном окне. Кнопка “>” загружает список изображений на сервере. Кнопка “Загрузить случайную картинку” отображает произвольную из имеющихся картинок или если список доступных изображений пуст, пытается получить его с сервера. Кнопка “Перемешать” перемешивает части изображения в соответствии с выбранной сложностью. Кнопка “Заново” сбрасывает таймер и отображает первоначальное состояние игры (или пустой виджет посередине с кусочками слева для классического пазла, или полностью собранное изображение для n-Puzzle).



Таймер.

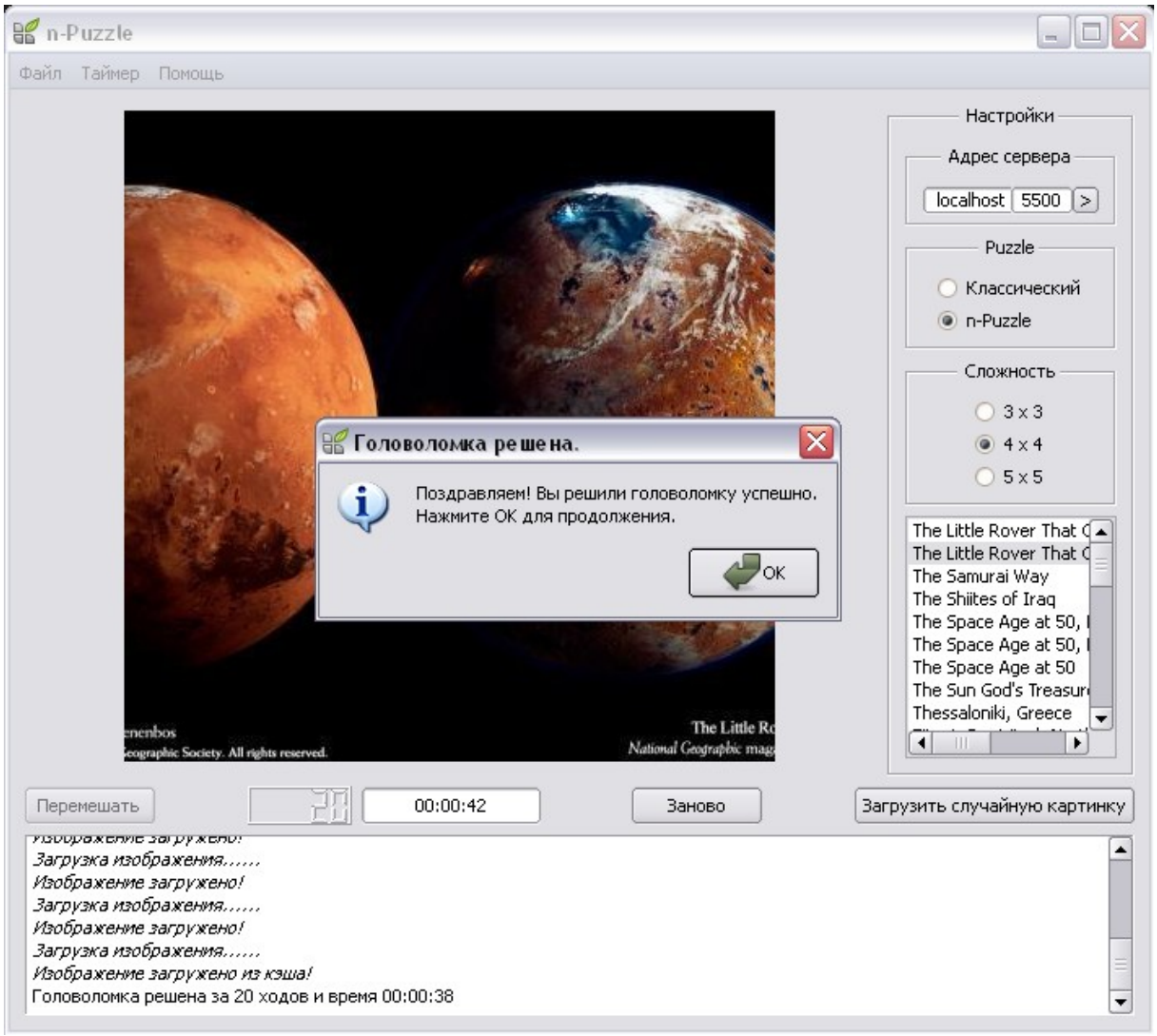
Присутствует также таймер. Таймер сбрасывается при нажатии кнопки “Заново” и начитает тикать сразу после нажатия кнопки ”Перемешать”. Таймер может быть включен или выключен, используя главное меню, и сброшен с использованием пункта “Таймер”->”Сбросить”.

Счетчик ходов.

Счетчик ходов используется для подсчета сделанный пользователем действий по перемещению отдельных элементов, в тоже время есть свои особенности. Так, например, засчитываются перемещение элемента в режиме “n-Puzzle” и перемещение отдельной картинки из левой игровой области в правую в режиме “Классический”, однако, перетаскивание изображения из правой области в левую оставит счетчик ходов без изменений.

Открытие/Сохранение изображений.

Открытие/Сохранение изображений реализованы через сконфигурированные диалоги по открытию/загрузке. Это означает, что диалоги настроены таким образом, чтобы сохранять/загружать только файлы картинок.



Виджет с логом.

Виджет с логом располагается как бы отдельно от главной игровой области. В этой виджете, которые иметь функцию прокрутки, отображается информация о различных событиях, которые происходят в программе. Например, когда картинка загружена, в это окно загружается текст, говорящий пользователю, что загрузка прошла успешно. При щелчке не кнопки “Перемешать” и “Заново” в этом виджете появится соответствующая информация. А также любая другая информация, которую следует видеть пользователю в процессе игры, может отображаться здесь. Замечание: виджет является не редактируемым. Представляется как дочерний виджет с автоматической прокруткой содержимого.

Сложность.

Группа переключателей, определяющих на какое количество кусочков будет разделено исходное изображение: 9, 16, 25.

Настройки сервера.

При загрузке списка картинок и непосредственно самих картинок используется механизм сокетов. Все требуемые файлы размещаются на сервере и клиентское приложение использует сокеты для соединения с сервером, выполняя соответствующие запросы и принимая ответы. Для настройки подключения к серверу используется группа элементов управления. С их помощью задается имя сервера и порт. При старте приложения происходит попытка получения списка изображений. В случае отрицательного результата эта группа элементов управления становится видимой. Отображение настроек сервера также зависит от того, выбран ли пункт меню “Файл”->”Настройки сервера”.

Изменение главного окна.

Главное окно приложение может изменяться по размеру. При этом происходит полное (в зависимости от размеров окна) или частичное (до некоторый пределов) деформация и/или изменение местоположения элементов управления. Область, содержащая картинку, не изменяет своего размера, но в тоже время ее положение меняется для сохранения пропорций и равномерного распределения появляющегося при изменении размеров окна свободного пространства.

Локальный кэш.

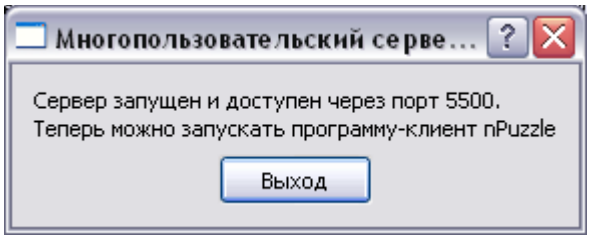
Во время работы приложение кэширует полученные с сервера изображения в папке “cache”. При следующем обращении к ранее загруженному изображению используется кэшированная картинка. При загрузке изображения, проверяется имеет ли оно размер 400x400 и формат jpg. В случае отрицательного результата на первую проверку изображение подгоняется под необходимые размеры, а файл с картинкой обновляется; на вторую – данное изображение не отображается и вместо него показывается стандартная картинка. При невозможности соединиться с сервером в списке изображений окна приложения отображается содержимое кеша. Поэтому папку локального кеша можно использовать и как хранилище файлов, используемых при игре. Картинка при этом должна быть формата jpg и может иметь произвольный размер.

2.3. Сервер

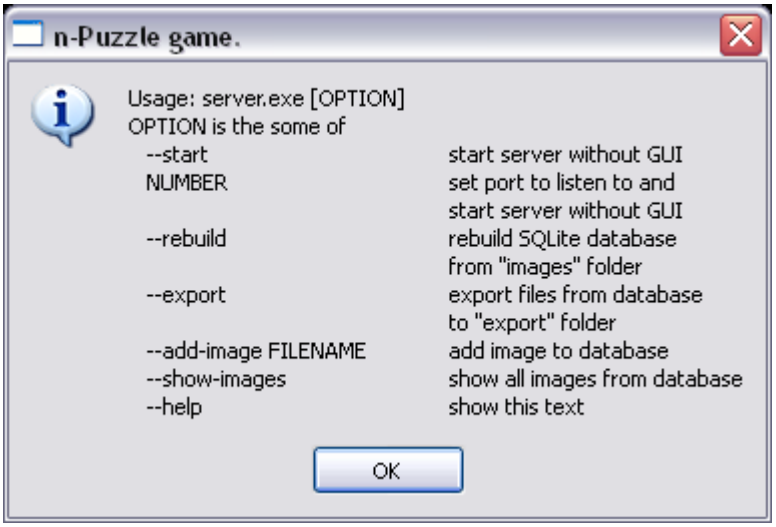
Программа сервер выполняет две основные задачи:

- администрирование базы данных с изображениями;
- ответы на запросы пользователей на получение списка доступных изображений и отправление самих картинок клиентским программам.

При запуске приложение-сервер прослушивает 5500 порт. При запуске приложения без параметров отображается графический интерфейс представленный ниже.



Информация (справка) о различных опциях работы с сервером становится доступной, если вызвать приложение с параметров “--help”. Тогда отображается информационное окно, которое можно видеть ниже (представлен случай запуска приложения из командной строки в Windows).



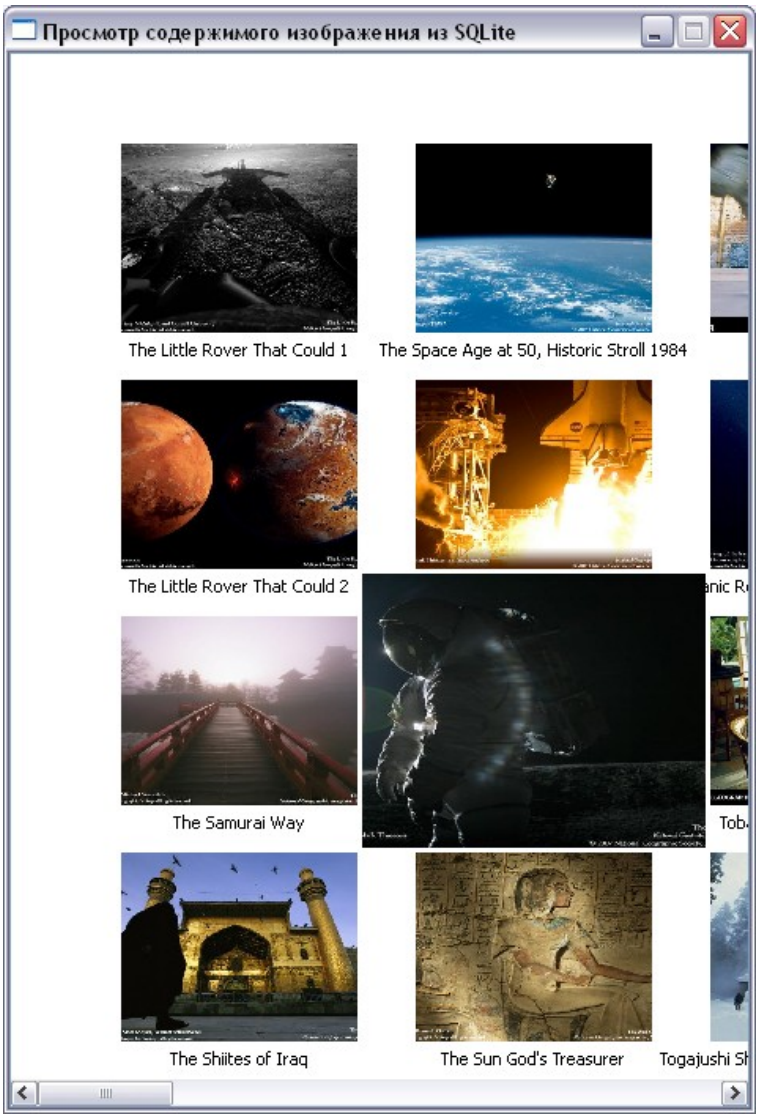
Изображения, предоставляемые сервером, хранятся в базе данных SQLite. При необходимости СУБД может быть изменена на MySQL, PostgreSQL, MS SQL Server, Oracle, IBM DB2, Sybase, Borland InterBase. Выбор SQLite связан с тем, что возможностей данной СУБД более чем достаточно для удовлетворения потребностей по хранению и предоставлению данных конечному пользователю. В тоже время использование СУБД дает серьезный задел на будущее. Так, например, в случае значительного увеличения числа файлов, предоставляемых сервером клиентам.

Таким образом, данное приложение-сервер является промежуточным звеном между клиентами, распределенными по сети и СУБД, хранящей различные изображения, т.е. выступает в роли сервера приложений в трехзвенной архитектуре.

Функции по администрированию изображений в базе данных включают следующие возможности:

- пересборка (опция --rebuild)
- извлечение (опция --export)
- добавление одной картинки (опция --add-image)
- отображение всех доступных пользователю изображений (опция --show-images)

В случае вызова приложения с опцией --show-images пользователь увидет окно, представленное ниже.



В базе данные файлы изображений хранятся в подготовленном к пересылке клиенту виде, что сокращает время отклика на запросы клиентов путем увеличения быстродейсвию приложения сервера. Заполнить базу данных картинками можно двумя способами:

- по одной картинке, вызывая приложение-сервер с параметром --add-image с указанием имени файла.
- создать в папке, содержащей исполняемый модуль приложения-сервера, вложенную папку “images”. Далее поместить в последнюю картинки, которые хотим сделать доступными для клиентов. После этого вызвать приложение-сервер с опцией --rebuild. Произойдет удаление имеющихся данных и добавление в базу новых картинок.

В процессе обновления базы изображения подвергаются ряду измений: преобразование в формат jpg, изменение размеров до 400x400 пикселей путем масштабирования и обрезания лишних частей изображения. Сохранность данных во время изменения данных обеспечивается принципами, лежащими в основе работы самой базы данных SQLite и путем использования механизма транзакций. Другими словами с одной и той же базой данный может работь несколько серверов приложений, предоставляющих доступ к изображениям через различные порты.

В базе данных для выполнения задач приложения-сервера создается одна таблица, сведения о полях которой приведены в таблице 6.

Таблица 6.

Имя поля	**Тип**	**PK**	**Описание**
id	INTEGER	Да, по возрастанию	Идентификатор изображения
name	TEXT		Базовая часть имени (без расширения) картинки
imagedata	BLOB		Массив байт, представляющий из себе изображение в формате jpg.

Как видно из таблицы 6 в базе данных SQLite используются типы INTEGER, TEXT, BLOB. В дополнение к этим типам также есть возможность создавать поля с типами REAL и NULL.

При взаимодействии с пользователем сервер принимает через сеть всего одно поле размером 32 бита, содержащее информацию о типе требуемой информации. Информацией может быть либо список файлов, хранящийся в базе данных, либо идентификатор (Идентификатор изображения в базе данных). В ответ в зависимости от типа данных, неоходимый клиенту, отправляется набор полей с указанием типа обработанного сообщения, размер полезных данных и непосредственно сами данные: набор строк или массив байт.

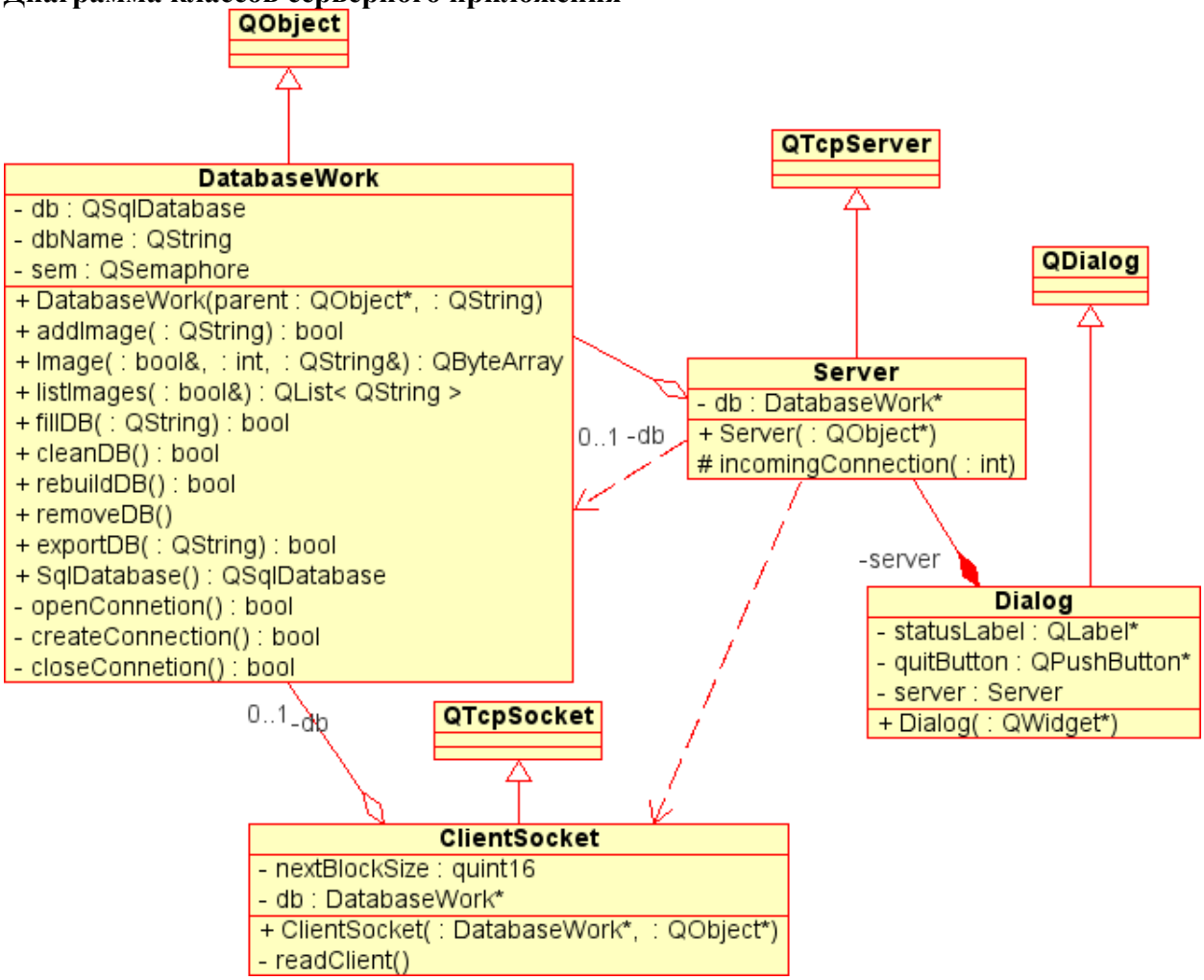
Сервер является многопользовательским приложением. Однако, для обеспечения возможности одновременной работы приложения с несколькими пользователями, программисту не обязательно организовывать работу нескольких потоков. Данная возможность связана с особенностями архитектуры приложений, построенных на основе фреймворка Qt.

Корректная работа с базой данных нескольких параллельно запущенных потоков в пределах одного процесса обеспечивается использованием объекта синхронизации, мьютекса, и классов, входящих в фреймворк Qt по работе с ним.

Таким образом, приложение-сервер является многопоточным и многопользовательским и выступает в роли сервера приложений в трехзвенной архитектуре. В месте с тем заложенные возможности по масштабированию позволят адаптировать приложение к работе с все более и более возрастающим количеством пользователей.

2.4. Диаграммы классов на языке UML

Диаграмма классов серверного приложения



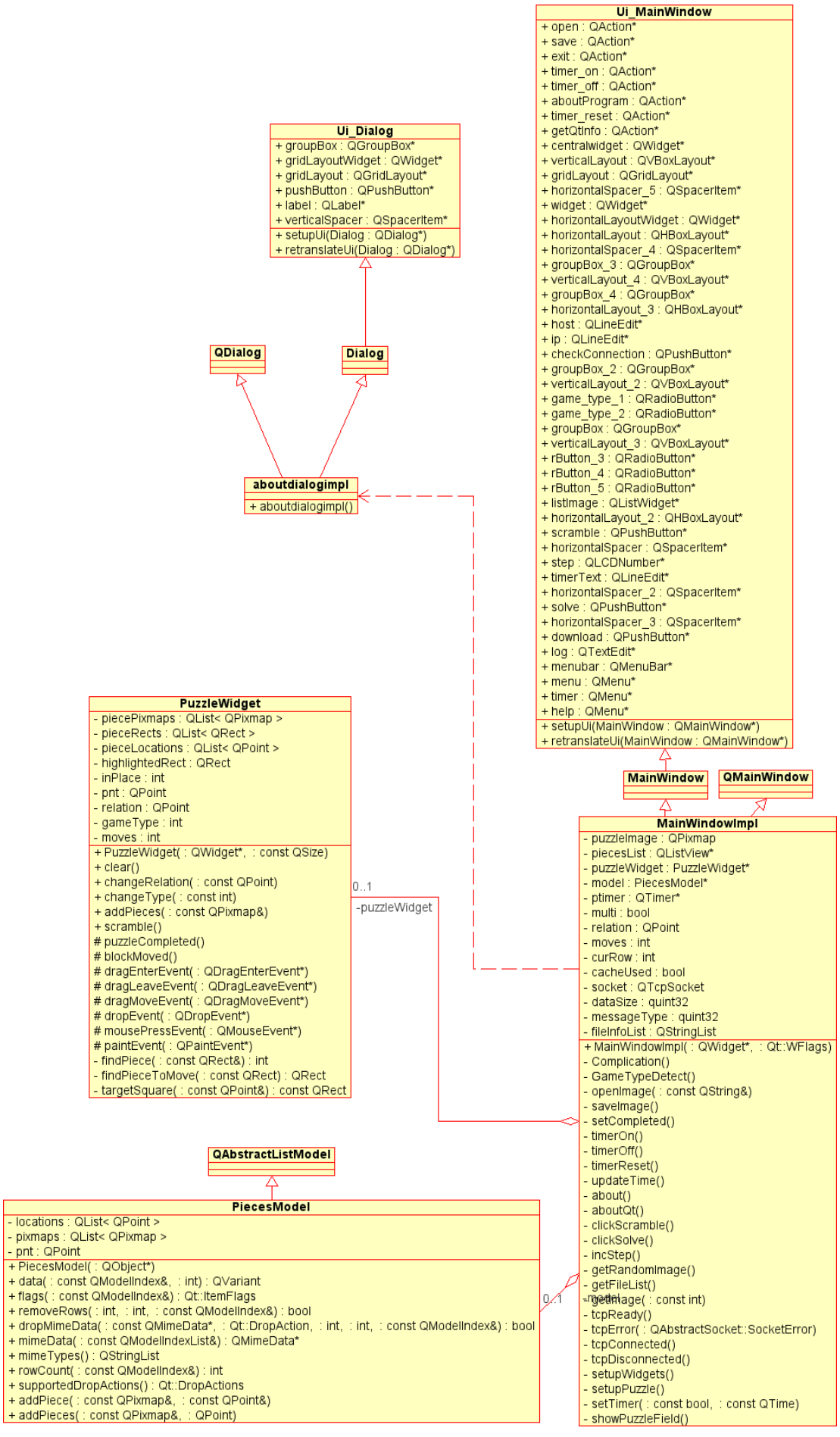
В зависимости от того, с какой опцией было запущено приложение, создается экземпляр класса DatabaseWork (инкапсулирует особенности работы с базой данных), Server (инкапсулирует работу по приему входящих соединений) или Dialog (служит для отображения кнопки для быстрого завершения приложения).

При запуске приложения без параметрос сначала создается экземпляр класса Dialog, далее он инстанцирует класс Server, настраивает его на обработку входящих соединения по заданному порту. Объект класса Server создает объект класса DataBaseWork и сохраняет указатель на него в скрытой переменной.

При появлении нового входящего соединения, просходит выхов защищенного метода incomingConnection(), в котором инстанцируется класс ClientSocket. Обработкой сообщений от него занимается главный поток. В конструкторе объекту класса ClientSocket передается указатель на заранее созданный объект класс DataBaseWork. Для исключения всевозможных недоразумения при работе нескольких поток одного процесса использует экземпляр класса QSemaphore.

Для корректной работы нескольких процессов с базой данных (например, при перестройке базы данных во время работы сервера с клиентами) используются средства синхронизации, предоставляемые конечной базой данных, в частности механизм транзакций.

Диаграмма ключевых классов клиентского приложения



Классы Ui_MainWindow и MainWindow, Ui_Dialog и Dialog создаются автоматически в процессе компиляции из mainwindow.ui и aboutDialog.ui соответственно. Файлы mainwindow.ui и aboutDialog.ui являются xml-документами. Класс QMainWindow несет в себе общие для всех форм базовые особенности поведения и является частью фреймворка. Класс Ui_MainWindow содержит только особенности данной конкретной формы, т.е. ее внешний вид и часть функциональности, отсутствующей в QMainWindow. Далее возможности этих двух классов объединяются посредством механизма множественного наследования в класс MainWindowImpl, где содержится реализация особенностей поведения формы при взаимодействии с пользователем. Аналогичные зависимости существуют и в группе классов Ui_Dialog, Dialog, QDialog и AboutDialogImpl.

Класс PuzzleWidget, наследник QWidget, является реализацией виджета, предназначенного для манипуляции кусочками изображения в процессе игры. Класс PiecesModel, наследник QAbstractListModel, является частью model фреймворка model-view-delegate (адаптация шаблона model-view-controller для фреймворка Qt). Часть View реализуется использованием готового класса QListView, который представляется в виде списка изображений, кусочков исходной картинки, которые можно переместить на виджет PuzzleWidget в режиме “Классический”. В режиме “n-Puzzle” виджет PuzzleWidget не отображается. Часть delegate в данном случае отсутствует.

Работа приложения состоит из создания экземпляра класса MainWindowImpl в функции main() приложения с дальнейшим отображением формы на экране монитора и запуска обработчика событий.

2.5. Кросс-платформенность

Игру n-Puzzle с полной уверенностью можно назвать представителем кроссплатформенного программного обеспечения. Или другими словами программного обеспечения работающего более чем на одной аппаратной и/или операционной платформе. Эта важная особенность позволяет данной реализации игры существовать в различных средах.

Как выяснилось в процессе портирования игры, данный процесс происходит очень легко. Единственная проблема, которая возникла при переносе кода, написанного под WindowsXP в Linux, - несоответствие кодировок, принятый по умолчанию в данных операционных системах. А если представить, что реальные проекты могут состоять из нескольких десятков файлов с исходными кодами, то требуется способ безболезненного обхода данной проблемы. Такой способ существует. Для этого можно использовать следующую строку, помещенную в файл main.cpp:

```
QTextCodec::setCodecForTr(QTextCodec::codecForName("CP1251"));
```

Также следует любой набор символов, используемых, например, для задания заголовка диалогового окна, стандартных диалогов и т.д., заключать в статический метод QObject::tr(). Это будет также полезно и в дальнейшем при сопровождении программы, например, во время локализации. Также будет не лишним познакомиться с советами по работе с фреймворком Qt, приведенными в приложении 1.

2.6. Вид окна при различных стилях оформления

Для окон/виджетов приложений, созданный при помощи кросс-платформенной библиотеки Qt, присутствует возможность задать стиль оформления элементов графического интерфейса. При явном задании стиля собранное из исходных кодов приложение будет выглядеть одинаково на всех поддерживаемых платформах. В случае, когда стиль нигде явно не определяется, используется стиль текущего окружения.

Библиотека Qt предоставляет следующие предопределенные стили:

- Cleanlooks (класс QCleanlooksStyle);
- Plastique (класс QPlastiqueStyle);
- Windows (класс QWindowsStyle);
- WindowsXP (класс QWindowsXPStyle);
- WindowsVista (класса QWindowsVistaStyle);
- Motif (класс QMotifStyle);
- CDE (класс QCDEStyle);
- Mac (класс QMacStyle, доступен только на платформе MacOS);

Стиль приложения задается в файле main.cpp:

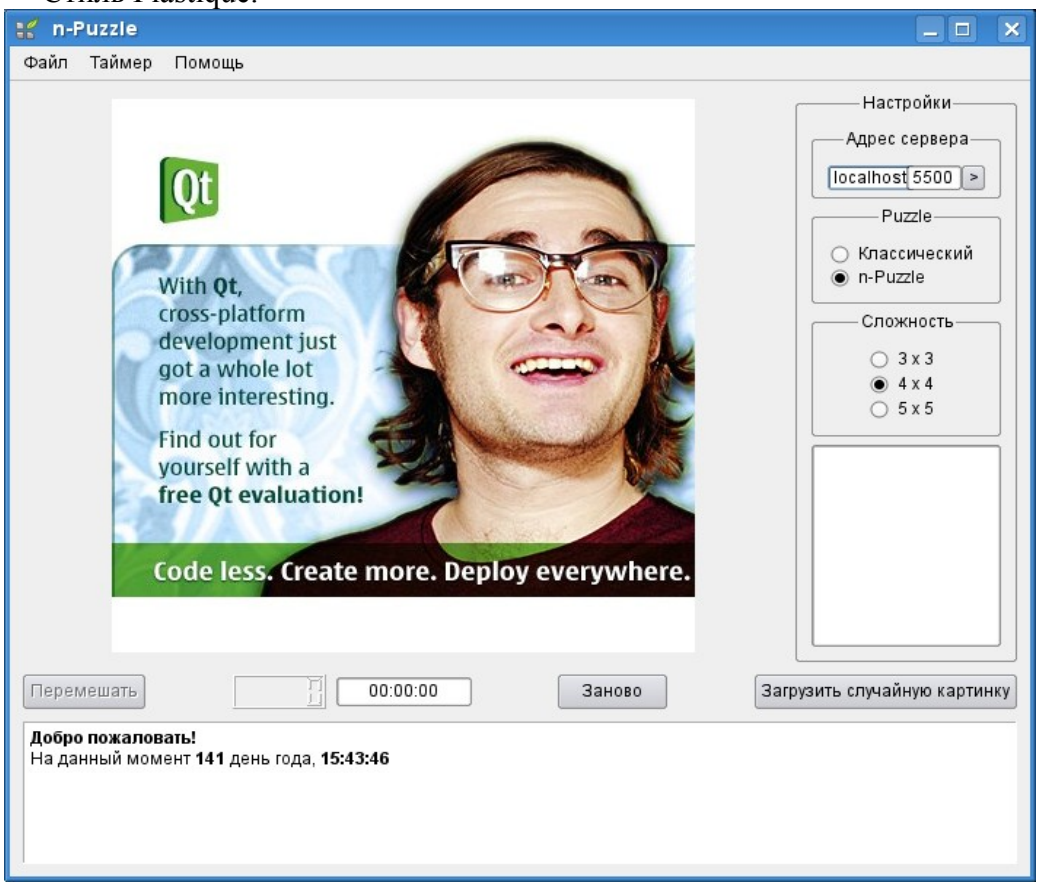
```
#include <QStyle>
#include <QCleanlooksStyle>
 int main(int argc, char ** argv)
{
        …
        QStyle *style = new QCleanlooksStyle();
        app.setStyle(style);
        …
}
```

В то же время доступна возможность создания собственных стилей оформления.

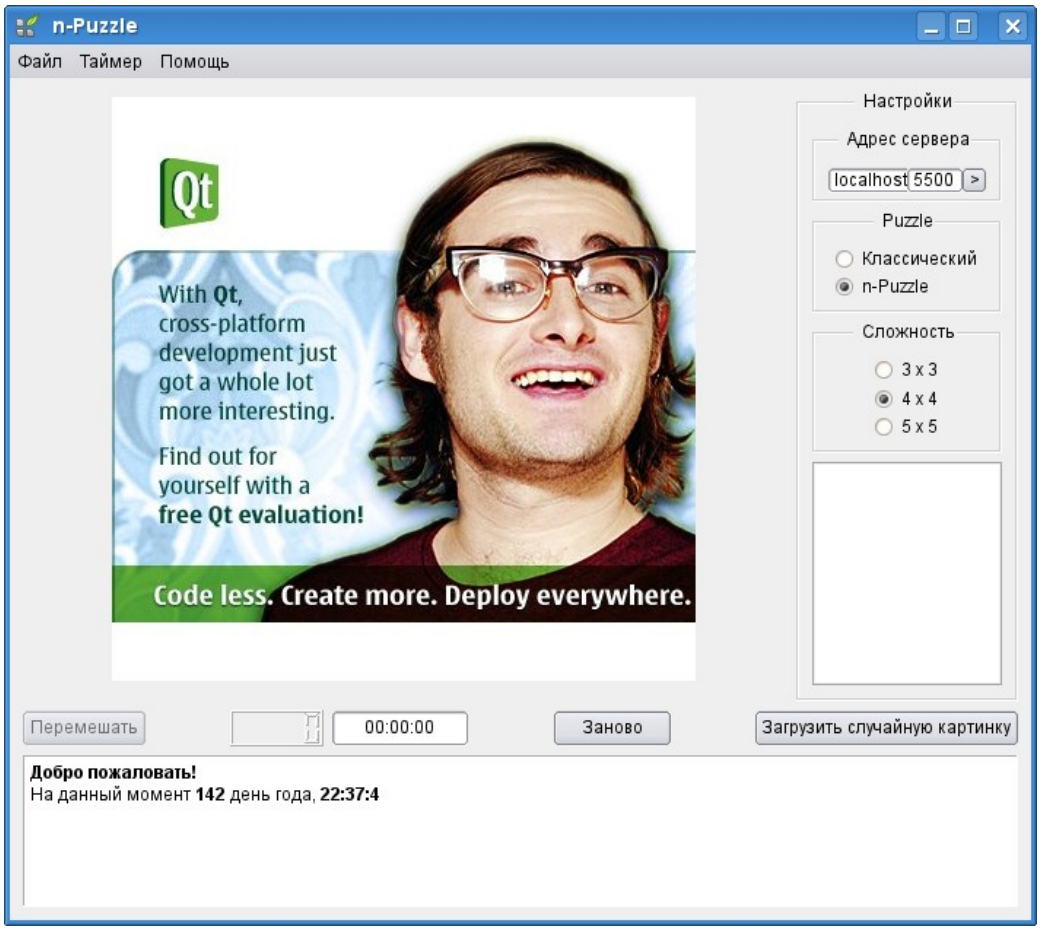
Ниже приведен вид окна в WindowsXP, если стиль явно не задан.

Стиль Cleanlooks.

Вид окна в Linux при различных стилях оформления окон.
Стиль Plastique.



Стиль Cleanlooks.



2.7. Стиль оформления кода приложения

C++ является мощным языком, поддерживающим множество различных стилей программирования. Стиль программирования, используемый в большинстве программа Qt, не является чистым стилем C++. Используется комбинация макросов и директив препроцессора для того, чтобы стать высокоуровневым языком более близким к Java и Python, чем к C++.

Наиболее важными особенности стиля программирования, которого придерживаются разработчики, использующие Qt, представлены ниже:

- имена классов начинаются с заглавной буквы (например, class Customer);
- имена функций начинаются с маленькой буквы;
- исключить где это возможно (за исключением случаем, рассмотренных ниже) подчеркивания, тире и “смешные” символы;
- в именах из нескольких слов каждое новое слова начинается с заглавной буквы (например, class FileTagger, void getStudentInfo());
- Константы должны состоят только из заглавных букв;
- имя каждого класса должно быть существительным или фразой, полученной из существительных (например, class LargeFurryMammal);
- каждое имя функции должно быть либо глаголом, либо глагольной фразой (например, processBookOrder());
- имя каждой переменной типа bool должно быть адаптировано для случаев использования в выражении if() (например, bool isQualified).

Для членов классов в большинстве случаев используются префиксы, хотя это не является обязательным и носит рекомендательный характер:

- имена членов классов: m_Color, m_Width;
- имена статических членов классов: sm_Singleton, sm_ObjCount.

Однако два последних правила часто не соблюдаются. Например, тем же дизайнером форм. Большинство примеров, рассмотренных мной, также придерживаются другого правила: имена членов классов начинается с маленькой буквы.

Для каждого атрибута, следует придерживаться определенных правил именования в функциях получения и задания значений (get, set):

- для функций, используемых для получения переменных всех типов, за исключением bool: color() или getColor();
- для функций, используемых для получения переменных типа bool: isChecked() или isValid();
- для функций, используемых для установки определенного значения: setColor(const Color& newColor).

Приведенные выше правила именования существенно улучшают читаемость и удобство дальнейшего их использования в программах.

Заключение

Данная учебно-исследовательская работа посвящена изучению фреймворка Qt и разработке с его помощью игры n-Puzzle, включающей в себя клиентскую часть, сервер приложений и базу данных SQLite в качестве места для хранения картинок.

В ходе выполнения работы достигнуты следующие результаты:

- Изучены:
  - платформа Qt;
  - средства создания GUI-интерфейсов и связанные технологии;
  - средства Qt для взаимодействия приложений через сеть на основе сокетов;
- Разработана игра n-Puzzle.
- Проверена работоспособность приложения в условиях различных операционных систем.

Разработка игры n-Puzzle позволила закрепить на практике полученные в ходе учебно-исследовательской работы знания и выработать навыки по работе с фрейворком Qt: набор классов и утилит. Разработка проводилась в различных операционных системах: то в OpenSuse 11.1, то в Windows XP. Применялись кросс-платформенные среды разработки: Qt Creator 1.0.0 и QDevelop 0.27. Это дало возможность ощутить преимущества и недостатки работы не только различных IDE, но и познакомиться с другими инструментами для разработки, поддерживаемыми различными операционными системами.

Литература

1. Ж. Бланшет, М. Саммерфилд Qt 4: Программирование GUI на C++. — М.: «КУДИЦ-ПРЕСС», 2007.— ISBN 978-5-91136-038-2

2. Макс. Шлее Qt 4: Профессиональное программирование на C++. — СПб.: «БХВ-Петербург», 2007.— ISBN 978-5-9775-0010-6

3. Alan Ezust, Paul Ezust. An Introduction to Design Patterns in C++ with Qt 4 – Prentice Hall, 2006. – ISBN 978-0-13-187905-8

4. Molkentin, Daniel. The Book of Qt 4: The Art of Building Qt Applications. – No Starch Press, Inc., 2006. – ISBN 978-3-937514-12-3

5. Thelin, Johan. Foundation of Qt Development – Apress,2007 – ISBN 978-1-59059-831-3

6. Trolltech Qt 4.5 Whitepaper – Trolltech ASA, 2006.

Приложение 1. Приемы программирования с Qt

(перевод материалов с сайта www.ics.com)

Просмотрите примеры, демонстрации, готовые решения от Trolltech, партнеров и проекты с открытым исходным кодом

- Нечто похожее на то, что нужно в данном случае уже существует
- Примеры Qt писались с особой тщательностью для того, чтобы показать хорошие практики программирования
- Проверьте условия лицензии до того, как использовать существующий код
- Исследуйте дополнительные модули фреймворка Qt
- Часть модулей GUI обеспечивает независимость от платформы для ключевых технологий: потоки, сеть, XML, Graphics View, OpenGL, SQL и многие другие…
- Что необходимо вашей программе, возможно, уже реализовано в этих модулях

Узнайте больше о qmake

- Qmake может упростить процесс сборки для кроссплатформенной разработки
- Создает Makefile, nMake Makefiles,
- Создает файлы проектов для Visual C++, VS NET и MacOS Xcode
- Может иметь некоторые ограничения для крупномасштабных проектов

Создавайте QObject в куче, используя new с указанием родителя

- Создание QObject с использованием new защитит его от неожиданного удаления
- Object удалится при удалении его родителя
- Безусловно, все QObject'ы имеют родителя, что, в конечном счете, гарантирует удаление данного объекта

Не используйте множественное наследование от QObject и классов, уже наследованных от QObject

- Qt не поддерживает данную возможность
- Понимайте различие между QDialog::exec() и QWidget()::show()
- Эти два часто используемые метода имеют очень похожее предназначение и могут вызвать недоразумения при знакомстве с Qt
    - QWidget::show() отображает виджет и его дочерние виджеты
    - QDialog()::exec() отображает диалог и его дочерние виджеты, а также ожидает пока пользователь закроет его, при этом блокируется ввод во все другие виджеты приложения, другими словами виджет диалога становится модальным
- Знание различий между этими двумя методами дает понимание различия между QDialog и QWidget
- QWidget – базовый класс для объектов графического интерфейса
- QDialog получается в результате наследования от QWidget и используется для получения необходимой информации от пользователя для продолжения работы приложения

Узнайте больше и используйте классы контейнеров, доступных в Qt

- Библиотека Qt предоставляет набор шаблонных классов контейнеров
- Они включают: QList<T>, QVector<T>, QSet<T>, QStringList и др.
- Их использование делает взаимодействие с другими классами Qt более легким, а код более понятным

- Используется скрытое хранение данных, что улучшает использование памяти и общую производительность

До того, как приступить к реализации своего виджета, убедитесь, что такое виджет уже не существует.

- Все классы объектов графического интерфейса в Qt фактически уже являются таковыми
- Qt предлагает часто используемые виджеты: QCalendar, QTooltip, QTabWidget, QDateEdit, QTimeEdit и многие другие
- Реализация своего виджета потребует изменения QPaintEvent и использования QPainter
- Если интерфейс нового виджета отличается от общепринятого, то есть вероятность что этот виджет не сможет работать правильно

Предпочитайте использовать Layout'ы вместо задания позиций для элементов графического интерфейса

- Включение виджетов в Layout'ы способствует более удобной работе с графическим интерфейсом
- Layout'ы изменяют ваши дочерние виджеты, когда
  - Изменяются размеры
  - Изменяются шрифты
  - Изменяется текст
  - Одно из дочерних виджетов исчезает, становится невидимым

Когда используется Qt-Designer и создается библиотека, используйте технику делегирования, а не наследования

- Добавляется гибкость, что позволяет использовать различные класса графического интерфейса во время выполнения
- Как результат может немного ускорить компиляцию

Определитесь, каким образом ваше приложение будет реагировать на события, посылаемые графическим интерфейсом.

- Графические приложения управляются событиями
- Важно знать 3 способа взаимодействия классов с помощью событий
- Сигналы/слоты
  - Эффективно (часто одна строка кода), если и сигнал, и слот уже существуют
  - Требуется метаобъектная система
- Переопределение методов событий (QKeyPressEvent, QMousePressEvent, QFocusInEvent, QTimeEvent и др.)
  - Большое число способов обработки событий
  - Может быть реализовано только на виджетах, для которых программисту разрешено модифицировать код
- Установка фильтров событий
  - Требуется регистрация виджета с использованием "installEventFilter()" и переопределить bool QObject::eventFilter()
  - Быстрый способ получить ответ на события без использования механизма наследования

Узнайте больше и используйте все возможные классы Qt

- QFile для работы с файлами
- QPixmap, QImage для использования картинок в приложении

- QSettings для хранения настроек приложения
- QDesktopServices для использования сервисов, предоставляемых рабочим столом
- QRegExp для преобразования данных с использованием регулярных выражений

Используйте QString, а не char*, помещайте все строки в tr()
- Поддержка Unicode
- Обеспечивается поддержка локализации

Знайте, что классы Qt используют скрытое распределение памяти
- Скрытое распределение позволяет безопасно осуществлять копирование аргументов, не волнуясь о накладных расходах при копировании

При использовании виджетов для представления списков, предпочтительна подход на основе подхода модель/представление, как противоположность подходу на основе элементов
- Намного легче создать модель, чем копировать информацию в элементы, постоянно при этом синхронизировать
- Исключается дублирование данных
- Позволяет использовать повторно некоторую модель для нескольких представлений и наоборот

Узнайте больше о возможностях отладки в Qt
- qDebug() используется для вывода отладочной информации
- qWarning() используется для сообщения об ошибках
- qCritical() используется для сообщения о критических и системных ошибках
- qFatal() используется для сообщения о фатальных ошибках с последующим завершением работы программы

Узнайте больше о наиболее частых ошибках программирования с использованием Qt
- макрос Q_OBJECT является обязательным для всех объектов, реализующих сигналы, слоты или свойства
- забыв об этом, как результат – появление странных ошибок при линковке
- синтаксис команды connect допускать наличие ошибок при его написании

```
//Эти ошибки не будут определены во время компиляции
connect(&a, SIGNAL(valueChanged(int a_variable)), &b,SLOT(setValue(int
a_variable)));

//int setValue(const QString&);
connect (&a, SIGNAL(valuetChanged(int))),&b,SLOT(setValue(QString&))));
```

Используйте Qt-Linguist для перевода
- файлы ts можно легко повредить при непосредственно ручном редактировании
- также удостоверьтесь, что
  - все строки, которые видит пользователь, заключены в tr()
  - используйте layout'ы
  - загружайте перечень строк при старте
  - локализуйте даты, времена, валюты и др.
  - Сделайте это сейчас, а не когда потребуется версия приложения для поставки в Китай или Гибралтар.

Используйте листы рассылки Qt-interest или форум Qt Centre

- В листах рассылки Qt-interest происходит обсуждение большим количеством активных пользователей Qt (http://lists.trolltech.com/qt-interest/)
- Qt Centre – это сайт сообщества, нацеленный на программирование с Qt (http://qtcentre.org/)